MUMPS PROGRAMMERS' REFERENCE MANUAL

MUMPS 開発委員会 編

標準マンプス 言語マニュアル

名古屋大学医学部
若井一朗・田中豊穂・嶋　芳成 共訳

コロナ社

MUMPS 開発委員会 編

標準マンプス
言語マニュアル

名古屋大学医学部

講師 医学博士 若 井 一 朗

田 中 豊 穂 共訳

嶋 芳 成

コ ロ ナ 社

このマニュアルは

1975年3月12日付 MDC/28「MUMPS 言語標準仕様」

1976年3月9日付　同上　正誤表

（これらは National Bureau of Standards より
1976年1月6日付「NBS Handbook 118」の第1部
1976年3月6日付　同上　正誤表
としても発行された.）

によって定義された言語に基づいている.

またこの本は, 1976年8月31日改訂の同じ書名と文書番号 MDC 3/5 をもつ文書を 1977年3月25日 MUMPS 開発委員会が MDC 公式文書として認可したものである.

謝　　辞

この文書は米国標準局（National Bureau of Standards）の契約番号 5-35770 により Melvin E. Conway 氏によって作成された.

彼は米国標準局購入注文番号 512576 のもとにプログラミングの例を多く開発した Paul L. Egerman の助力を受けた. この研究は合衆国保健教育福祉省の Health Resources Administration の National Center for Health Services Research および合衆国商務省の米国標準局の Institute for Computer Science and Technology との共同協定によって委託されたものである.

邦 訳 補 遺

上記の NBS Handbook 118 は 1977年9月15日付 American National Standard MUMPS Language Standard, ANSI X11, 1-1977 として MUMPS 開発委員会から出版された.

日本語版への序文

ユーザー共同体がその言語仕様を受け入れたときに初めてその標準言語が現実に存在すると言えます．標準 MUMPS の存在はこのマニュアルに見られるような日本マンプス ユーザーズ グループの標準 MUMPS 実践への努力によって保証されています．

私は 1959 年よりプログラミングを手がけ，1970 年以来 MUMPS を使用してきましたが，それにより MUMPS の力量と言語標準化の必要性について充分に判断できるようになりました．幸いにもそのために私は標準 MUMPS の開発に参加でき，また標準言語仕様の開発過程で，MUMPS 開発委員会の委員長として，技術者とユーザーが話し合い一字一句合意に達してゆくのを見守るという，又とない機会に恵まれました．

このように比較的短い期間内に言語仕様が作成され認可されることができたのは，自ら莫大な能力と時間を割愛した委員会メンバー達の積極的な意志と NBS の支援による所が大でありました．“標準 MUMPS への努力”に参加した私達全員はその努力の結晶であるこのマニュアルの日本語による刊行をみたことを心から喜んでおります．

1977 年 10 月 5 日

第 1 期 MDC 委員長

R. Peter Ericson

日本語版への序文

MUMPS 開発委員会委員長として，この「標準マンプス言語マニュアル」の日本語出版を紹介する機会を得たことを喜んでいます．MUMPS 共同体はつとに，MUMPS への努力の全経過を通じて，日本の人々がこれを支持し，またこれに寄与して下さったことを認めております．この日本語訳が出版されることによって，標準 MUMPS 言語を利用することと，更にこれを発展させることへの共通の関心が強められんことを私達は期待しております．

1977 年 6 月 7 日

第 2 期 MDC 委員長

Jack Bowie, Sc.D.

日本語版への序文

「標準 MUMPS 言語マニュアル」の翻訳権許可による日本語版が，日本MUMPS ユーザーズ グループ関係者の手により出版されることは喜びにたえません．日本の MUMPS への努力が非常に強力でかつ多様な展開をみていることはつとに我々に感銘を与えるものでありました．事実私は 1976 年 10 月，日本 MUMPS ユーザーズ グループの会合に列席し MUMPS 言語の装備と利用法に関する日本の MUMPS 共同体の努力をまのあたりにし，同時に日本の病院や医学部でおこなわれる研究の幅広さについても以前からの印象を強めました．

もう一つ私の大きな喜びとしていることは，日本の主要な産業が「標準」言語の装備にはじめから目を向けていたことです．我々自身の国でさえこの分野では一部しか手をつけていない時に日本の主要コンピュータ企業の大半が「標準」言語を採用し，その利用への方針を発表していました．既に或る会社が米国製の機械を用いて「標準 MUMPS」の装備をはじめていたことは，日本の「標準」への強い決断を示していたのだといえます．今この「標準 MUMPS」は米国国家規格協会（ANSI）の認可を受け，米国の産業界が改めてこれを採用することになろうとしていますが，我々は日本の皆さんが最初に道を指し示したのだということを忘れることができません．

この言語参照マニュアルは，MUMPS 言語を装備する国の国語で読むことができればより有用な文書であり，技師や医師が自身のプログラムを書く時に，“ローマ字”一字とそれで書く応用プログラムとの間の橋渡しをするのに大いに役立つことになりましょう．

このマニュアルの日本語版が情報伝達の壁をいかに薄くするかを日本語初心者の私は認めない訳にはゆきません！

我々は，言語定義のイニシアティブをとってきたことを誇りとしておりますが，今後日本でMUMPSを使う人々が多くなり，各々の知識技術を生かして我々と共にこの言語を磨きあげる努力に参加されることを期待しております．本書と他の教科書が日本で出版されることで，我々両国の二つのユーザーズグループ間の情報伝達も促進され，言語の改良についての皆さんの提案をいただけるようになることを疑いません．皆さんの立派な業績を祝賀しそれがなお一層の隆盛に通じますことを念願致します．

1977年8月31日

第2期MUG会長

Richard F. Walters, Ph. D.

日本語版への序文

MUMPS とは元来ボストンのマサチューセッツ総合病院（MGH）で開発されたコンピュータ言語で，それはコンピュータの歴史の中でも特異的な進化の過程を踏みました．まず1966年に MGH Utility Multi-Programming System（MUMPS）として G.O. Barnett 博士や A.N. Pappalardo 氏らによって手がけられたときからその通訳語（インタプリタ）としての特徴と文字列操作やファイル操作の特性が認められて急速な普及をみました．

これは直ちに MGH の環境を離れて新版の展開をみるようになり，それらが原版の特性を変えて装備されたため，一連の‘MUMPS方言’が出現することとなりました．1972年までには6種の方言が医学や企業の分野で，教育用，研究用，産業用として広く使われるまでになっていました．

その年，保健・教育・福祉省の国立保健サービス研究センター（NCHSR）と商務省の米国標準局（NBS）が共同で次の2つの目標を達成するためのグループを支援することとなりました．その1つは MUMPS 開発委員会（MDC）であり，他の1つは MUMPS ユーザーズ グループ（MUG）でありました．MDC はそれまでの諸方言を使用しているユーザーらにも受け入れられた形での「標準 MUMPS」を定義する作業を委託され，MUG は様々な方法で MUMPS 言語の利用を促進する役割をもちました．1976年のはじめ MDC はその目標である「標準 MUMPS」の定義と採択を完了し，「標準 MUMPS」の重要性はその後米国国家規格協会（ANSI）がこれを認可することにより確立され，今年，FORTRAN，COBOL についで ANSI 認可による第3の標準言語となろうとしています．

標準 MUMPS は文字列型のデータを階層的に組織して集めその操作を容易にするために設計開発され，主な特徴をあげると次のようになります．

1. テキスト（文章型データ）操作能力．　これによって任意にデータのキーワード的な特定部分を検証したり，データ文字列の特定の位置に文字があるか，数字があるか，句読点があるか，というような形式についても容易に調べることができる．このような能力は新たに入力されたデータを受け入れるか拒絶するかという場合に非常に重要であり，入力された時点でのオンラインデータ収集に役立つ．他のテキスト処理能力としては，必要に応じてテキストの断片を１つに連結したり，逆に１つのテキストを断片に分ける能力がある．
2. 階層的ファイル構造．　ユーザーはこの構造を用いてデータファイルを自分の必要に応じて設計できる．この構造は大抵の記録システム（例えば医学的記録，財政録など）が実際上重要度や詳細さについてレベルを異にするので意義が大きい．すなわちデータの構造を本来の論理構造に擬し，有用なファイルを直ちに作ることができる．
3. 高水準性．　この性質は，この言語の構文規則と意味論がこの言語によって解きうるような問題を指向していることに由来し，逆にある特定コンピュータを指向していないという点にある．MUMPS は，その能力の範囲をすべて利用するには何ヵ月も必要とするであろうが，習得第１日目に既にかなり有用な実用プログラムが書けるほど，比較的短期の訓練で用いることができる．

MUMPS 言語は効率を上げるためそれ自身のオペレーティング システムをもつのが普通であり，ときには既存のオペレーティング システムに装備されることもあります．オペレーティング システムはそれが MUMPS 独自のものであれ，より汎用的なものであれ，MUMPS を特徴づけるような能力を備えるはずであります．そのうち最も重要なものは次の４つであると言わねばなりません．

1. 多数のユーザーが１つの機械で同時に MUMPS を使用することができ，他のユーザーの活動によって影響を与えられることが比較的少ない．

2. どのデータベースも他のユーザーの作業とは無関係に，許されたものは誰でもそれと対話し調べ，かつ内容を変更することができる．これを‘データベース シェアリング’という．
3. MUMPS は普通‘インタプリタ’として装備される．
4. MUMPS は‘オンライン’システムである．

1977 年 9 月 3 日

第1期 MUG 書記長

Joan Zimmerman, Ph. D.

訳 者 序 文

本書「標準マンプス言語マニュアル」は MUMPS 開発委員会（MUMPS Development Committee）刊行になる文書のうち「MUMPS 言語標準」（MUMPS Language Standard, ANSI X11.1-1977, MUMPS Development Committee, 1977）に次いで現在最も権威とされる「MUMPS Programmers' Reference Manual」の邦訳である.

訳者らは本書を日本の MUMPS 利用者および言語装備関係者に，既に世界の本流となっている「標準 MUMPS」の仕様とその構文規則および解釈法を普及させるために，解りやすいしかもできるだけ原文に忠実な邦訳書とすることに心がけた.

本書の出版にあたり，米国 MUG Executive Secretary である Dr. Joan Zimmerman，米国 MUG Chairman である Dr. Richard F. Walters，米国 MDC Chairman である Dr. Jack Bowie および NBS の Mr. Joseph T. O'Neill の支援と，日本マンプス ユーザーズ グループの会員諸氏からの多くの助言に負うところが大であったことを記して深甚の謝意を表する.

1977 年 9 月 29 日，ANSI による標準 MUMPS 認可の報をきゝつゝ.

若井一朗
田中豊穂
嶋芳成

目　　次

第I部　序　　論

第1章　この本の目的 …… 1

第2章　この本の形式の説明 …… 3

2.1　この本の区分 …… 3
2.2　技術用語概論 …… 3
2.3　構文仕様のためのメタ言語 …… 5
2.4　初歩的な構文規則上の定義 …… 8
2.5　移送基準 …… 8
2.6　例の利用法 …… 9

第II部　MUMPS システム モデル

第3章　MUMPS システム モデル …… 11

第4章　システム記憶域 …… 14

4.1　名前付きシステム記憶域 …… 14
4.2　Lock リスト P-ベクトル …… 17
4.3　Open リスト P-ベクトル …… 19
4.4　作業番号 P-ベクトル …… 20
4.5　時　　計 …… 21

第5章　パーティション記憶域 …… 22

5.1　名前付きパーティション記憶域 …… 22
5.2　略式グローバル指標 …… 22
5.3　テスト スイッチ …… 23
5.4　現用装置指定子 …… 24
5.5　現用装置水平カーソルおよび垂直カーソル …… 26

第6章　パーティション スタックと実行順序の制御 ……28

6.1　通常の実行順序 ……28
6.2　順序の変更 ……30
1.　HALT ……30
2.　QUIT (「For 範囲」スイッチ オフ) ……30
3.　IF, ELSE ……30
4.　DO ……30
5.　GOTO ……31
6.　XECUTE ……31
7.　FOR ……31
8.　間接指定 ……33

第III部　言　　語

第7章　ルーチンと行 ……35

第8章　命　　令 ……39

8.1　予備的定義 ……39
8.1.1　命令の構文規則 ……39
8.1.2　命令の定義の形式 ……40
8.1.3　詳　　解 ……40
8.1.3.1　命令語の短縮 ……40
8.1.3.2　命令語の後付け条件 ……41
8.1.3.3　多引数の並記 ……42
8.1.3.4　引数の後付け条件 ……43
8.1.3.5　引数間接指定 ……44
8.1.4　命令の定義に共通する要素 ……45
8.1.4.1　READ と WRITE の中の書式 ……45
8.1.4.2　LOCK, OPEN, READ の時間制限 ……46
8.2　命令の定義 ……49
8.2.1　BREAK ……49
8.2.2　CLOSE ……50
8.2.3　DO ……51
8.2.4　ELSE ……53
8.2.5　FOR ……54

8.2.6 GOTO ……60
8.2.7 HALT ……62
8.2.8 HANG ……63
8.2.9 IF ……64
8.2.10 KILL ……66
8.2.11 LOCK ……69
8.2.12 OPEN ……74
8.2.13 QUIT ……77
8.2.14 READ ……79
8.2.15 SET ……85
8.2.16 USE ……90
8.2.17 VIEW ……92
8.2.18 WRITE ……93
8.2.19 XECUTE ……96
8.2.20 Z ……98

第9章 特殊変数 ……99

9.1 $HOROLOG ……101
9.2 $IO ……102
9.3 $JOB ……103
9.4 $STORAGE ……104
9.5 $TEST ……106
9.6 $X ……107
9.7 $Y ……108
9.8 $Z ……109

第10章 関　数 ……110

10.1 概　要 ……110
10.1.1 位置番号の定義 ……113
10.2 関数の定義 ……114
10.2.1 $ASCII ……114
10.2.2 $CHAR ……115
10.2.3 $DATA ……116
10.2.4 $EXTRACT ……118
10.2.5 $FIND ……119
10.2.6 $JUSTIFY ……121
10.2.7 $LENGTH ……123

10.2.8　$NEXT ……… 124
10.2.9　$PIECE ……… 126
10.2.10 $RANDOM ……… 128
10.2.11 $SELECT ……… 129
10.2.12 $TEXT ……… 131
10.2.13 $VIEW ……… 133
10.2.14 $Z ……… 134

第11章　式 ……… 135

11.1　式を支配する一般的法則 ……… 135
11.1.1　式の構文規則 ……… 135
11.1.2　MUMPS 値 ……… 137
11.1.3　解 釈 演 算 ……… 139
11.1.3.1　解釈 Ie （指数解釈） ……… 143
11.1.3.2　解釈 Ien (指数-数解釈) ……… 143
11.1.3.3　解釈 Ini (数-整数解釈) ……… 144
11.1.3.4　解釈 Int (数-真偽値解釈) ……… 144
11.1.4　式値の計算 ……… 145
11.2　式 尾 の 型 ……… 147
11.2.1　無限定式尾 ……… 149
11.2.2　数値限定式尾 ……… 150
11.2.3　整数値限定式尾 ……… 152
11.2.4　真偽値限定式尾 ……… 153
11.2.4.1　関係演算子 ……… 153
11.2.4.2　論理演算子 ……… 154
11.2.4.3　パターン照合演算子 ……… 155
11.2.4.4　真偽値演算子の例 ……… 157

第12章　式　　子 ……… 159

12.1　式子の構文規則 ……… 159
12.1.1　式子形成のための括弧の使用 ……… 159
12.2　定 文 字 列 ……… 160
12.3　数　字　列 ……… 160
12.4　記憶域参照 ……… 162
12.5　一項演算子 ……… 166

付　録

A　ASCII 文字表 …… 169
B　索　引 …… 170
C　MUMPS 開発委員会文書 …… 173

邦訳付録

翻訳にあたって用いた MUMPS の文字列の分類と訳語 …… 174

訳者紹介

若井一朗

1956年　名古屋大学医学部卒
1957年　Fulbright 留学麻酔学専攻
1962年　Fellow of American College of Anesthesiology
1963年　名古屋大学大学院医学研究科外科学専攻修了
1964年　医学博士
1968年　鳥取大学助教授
1969年　同大付属病院中央手術部長
1973年　社会保険中京病院麻酔科部長兼コンピュータセンター長
1975年　日本 MUG 発起人，米国 MUG 運営委員
1976年　マンプス システム研究所所長　同付属クリニック院長
1977年　名古屋大学医学部講師

田中豊穂

1967年　名古屋大学医学部卒
1972年　名古屋大学医学部助手
専攻：衛生学，環境衛生，母子保健

嶋　芳成

1973年　愛知県立旭丘高校卒　名古屋大学医学部医学科在学

第 I 部

序　論

第 1 章
この本の目的

この本は標準 MUMPS 言語を用いて応用プログラムを書く人たちのための参考書である．これは MUMPS 言語の既存の装備（implementation）とは無関係に書かれている．しかし，多くの言語装備にはそれぞれプログラマー参考書がついているはずであるから，この本のような特定の言語装備とは無関係な参考書のもつ役割はどのようなものかという疑問が当然起こるであろう．標準 MUMPS 言語の教科書としては既に『MUMPS 入門書』[訳注1)]があるので，この疑問は更に強くなる．しかしこの本は既に標準 MUMPS 言語を充分知っているプログラマーが応用プログラムを書くときに生ずる諸疑問を解く目的で書かれたものである．このようなプログラマーは次の要領でこの本を利用するのがよい．

1. この本は参照用であって，一たんこの本の使用法を理解するならば（第２章参照），あとは特定の疑問に対する特定の解答を得るのに役立つものとなろう．第２章を読んだのち，第３章以降の利用法が分かれば読者はこの本の適当な箇所を必要に応じて調べ，特定の疑問を解くことができるはずである．

2. この本は言語の構造よりもむしろ解釈上の疑問に力点を置いている．構文規則についても記述されているが，とくに“どんなとき何が起こるか？”という形の疑問に答えることを主眼とした．そのためこの本は多くの場合第３章の初めに述べられる“MUMPS システム モデル”が具体的にどのように動くかを示す形で説明している．この MUMPS シス

訳注 1) M.E. Johnson and R.E. Dayhoff, MUMPS Primer, MDC 1/11, 6/13/75.

テム モデルは，この本の解説目的のためにのみ案出された標準 MUMPS の仮想上の言語装備として見ていただきたい．実際の言語装備はこのシステム モデルの内部特性をそのまま模倣したものであるとは限らない．しかしながら，MUMPS 移送基準を遵守した MUMPS ルーチンならば，適正に装備された標準 MUMPS 言語と MUMPS システム モデルは同一入力に対しては互いに全く同一の出力をもたらすはずである．

3. この本では頻繁に参照し，利用し，容易に理解できる部分と同様に，言語の深遠な部分にも光を当てようと努力してある．

4. この本は『MUMPS 移送基準』[訳注1] を採用することによって，プログラムの機種間での移送性を高めようとしている．この移送基準は単に採用したのみでなく，それを明確に引用してある．『移送基準』を明示した理由はこれらの制約をやわらげている言語装備があった場合，この本の記述がその仕様と矛盾するように見えることを避けるためである．このような方法をとることで，ある種の言語装備が提供するかも知れない移送性以外の利点とこの移送性との間のバランスについては，その責任の所在すなわちプログラム活動の管理者にゆだねることにした．

この本は二義的には，MUMPS 言語装備者のためにも書かれている．言語装備者は言語の設計者たちが予期していなかったような疑問にも答えられるように，いつも言語の仕様を解釈し得なければならない．この本はできるだけ多くの疑問を想定しようと試みている．想定から洩れてしまった疑問とその解答を補うためにも，この包括的なモデルを採用することで，異なった言語装備者たちでも同様な解釈に収束できるような筋道の通った構造を示そうと努めている．このような目標は移送性の問題に大きな注意を払ってきた MUMPS の設計者たちにとっては特に重要な課題なのである．

訳注 1)　MUMPS Language Standard, ANSI X11.1-1977, MUMPS Development Committee, Par II, 1977.

第 2 章
この本の形式の説明

2.1 この本の区分

目次で示したように，この本は3つの部分に分かれている．

第Ⅰ部（第1～2章）は序論的な部分である．

第Ⅱ部（第3～6章）は MUMPS システム モデルを示している．第Ⅱ部では，後に言語の説明に使用される種々の概念が紹介されている．このために第Ⅲ部の前後参照の多くは，第Ⅱ部にもどって参照されることになる．

第Ⅲ部（第7～12章）では一般に漸次細かくルーチン(第7章)から式子（第12章）までにわたり MUMPS 言語の要素の構文規則と意味論を解説する．

以上のほかに3つの付録がある．

付録Aは ASCII 文字表であり，MUMPS プログラマーにとってこの表は文字の符号を10進法で表現している点に利用価値がある．

2.2 技術用語概論

この本で使われる技術用語は主に2つの種類に分かれる．

<u>構文規則上の用語</u>

構文規則上の技術用語は MUMPS システムで使われる文字列の種類に対して与えられる名称である．

それが何であるかは，その文字列のつづりによって定義されることになる．

重要なことは，構文規則上の技術用語として分類される文字列も，文脈の相違によって数種の型が生じる点である．

1. MUMPS ルーチンの部分として分類されるもの，例えばライン，命令，式，定文字列である．
2. MUMPS 処理過程によって操作を受けるデータについて分類されるもの，例えば数値，負でない整数値，真偽値である．
3. システム モデルで定義された過程によって処理された値として生じるもので，必ずしもデータとして取り出しうるとは限らないもの．例えば節指定子，装置指定子である．

このようにこの本で使用される構文規則の定義方式と使用方式は，ルーチンの文字列にもデータの文字列にも一律にあてはまるものである．

構文規則上の技術用語は常に全英つづりが小文字（時にはハイフン“-”で結合）でつづられる．文脈によっては，これらに下線をつけることがある．この下線は用語の性質を変えないで読みやすくする助けとしての意味しかもたない．

1. 構文規則上の技術用語が，構文規則の定義として使われているときは，常に下線がついている．この下線はこの用語の名称が複数の単語からなるとき特に重要である．従って式子（expression atom）とすれば，それが単一のものの名称であることが明らかとなる．

2. 文中で時に構文規則上の技術用語に下線をつけて，文脈上特定の語を参照しているのを明らかにすることがある．例えば SET 命令（8.2.15）の構文規則の記述では式という用語が＝符号の右に書かれている．そしてこの命令の実行に関する説明文のずっと下の部分に，文章で“式が評価される”と書かれている．ここで下線の使用は，これが構文規則の記述で用いられた式に対するものであることを明らかにしている．

p85

システム モデル上の技術用語[訳注1)]

「MUMPS システム モデル」(MUMPS System Model) の説明中，いくつかの技術用語が出てくる．ここでは「システム モデル上の技術用語」[訳注1)]と呼ぶことにする．このようにしたのは構文規則上の技術用語の種類とはっきり区別するためである．例えば「名前付きシステム記憶域」(Named System Storage),「作業番号 P-ベクトル」(Job Number P-vector),「時計レジスター」(Clock Register),「テスト スイッチ」(Test Switch),「現用装置指定子」(Current Device Designator),「略式グローバル指標」(Naked Indicator) などがある．

2.3 構文仕様のためのメタ言語

構文規則の仕様はすべての文字列を２つの種類，すなわちこの構文仕様に合致するものと，そうでないものとに分類する法則である．構文仕様は２つの文脈の中で生じる．

1. 構文規則の定義の右側に関するもの．構文規則の定義は３つの部分をもち，左側は定義される構文規則上の技術用語の名称であり，中間にメタ言語演算子 ::=，そして右側に構文仕様が来る．

   例えば第７章に次のようなものがある．

   行 ::= 行頭 行体
   行頭 ::= 〔ラベル〕1s
   行体 ::= ⎡命令〔␣命令〕 . . . 〔␣命令〕⎤ eol
   　　　　 ⎣注釈文⎦

   最初の例で定義した構文規則上の用語は行である．そして ::= は定義により構文仕様上，行頭，行体を満足させるいかなる文字列でも，構文規則上の行であることを示す．構文規則上の用語は一度は構文規則の

訳注 1) 原著では「頭文字が大文字の技術用語」(Capitalized Technical Terms) であるが邦訳では「 」を付し最初および適当な箇所で原語を（ ）内に示す．

定義の左側に現われ，時には繰り返されることもある．英字，文字と符号，引用符号以外の文字と符号のように，構文規則上の用語ではあるが，構文規則の左側に現われないものは，略式に定義されており，この本で出てくる場合には，必ずしもその都度ではないが1文字の文字列として定義される．

2. 自立型の構文仕様として，例えば構文規則上の用語 命令 は，正式にはどこにも定義されていないが，略式に定義するならば，命令は8.2で解説される個々の命令の構文規則のいずれによっても満足されるものとしてよい．例えば SET 命令は，次のような（簡略化した）構文仕様となる．

SET␣記憶域参照　=　式

構文仕様では，次のようなメタ言語の操作が生じる．

順　序

例：　行頭　行体

説明：　1.　行頭を満足させる文字列を左側に，

2.　行体を満足させる文字列を右側にして

これらを結合させたものとして表現できる文字列は，すべて 行頭 行体 を満足する．

選　択

例：| 式子 |
| 式　　式尾 |

説明：　縦線は問題となる文字列の境界を定義し，それ以外のメタ言語上の意義はない．時には（前述の行体の定義のように）この境界の定義機能を角括弧〔　〕でもたせるが，そのときは更に以下に述べるようなメタ言語上の機能をあわせもつ．

式子という構文仕様，ないし 式 式尾という構文仕様のどちらかを満足する文字はすべて

| 式子 |
| 式 式尾 |

という構文仕様を満足する.

任 意

例: 〔ラベル〕1s

説明: 〔 〕に囲まれたものは，あってもなくてもよい．この例で言えば，もし文字列がラベル 1sを満足しても，1sを満足しても，どちらも〔ラベル〕1sを満足する.

反 復

例: 命令|␣命令| . . . eol

説明: 3つの点...はその直前に先行する最小単位の構文仕様の任意の無制限な反復（この場合は␣命令であるが）を意味する.

従って次のどれかを満足させる文字列はこの例の仕様を満足する.

命令␣命令 eol

命令␣命令␣命令 eol

命令␣命令␣命令␣命令 eol

等々.

値の指定

例: @式子 V グローバル変数

説明: メタ言語演算子Vはその左の式子にも，またそのすぐ右の最小単位の構文仕様にも適用される．従って次のような意味となる.

1. この構文仕様の構文はまず@式子を満足し，そして
2. 式子が評価されたあと，その値の構文がグローバル変数を満足しなければならない.

注: @式子 V は間接指定の機会があることを示すために用いられている．このような構文仕様を伴う記述はすべて，間接指定が呼び出したものの評価が完了して生じた文字列に適用される.

2.4　初歩的な構文規則上の定義

原則的に次の構文規則上の定義が用いられる．

文字と符号　::=　95個の ASCII の文字と符号からとった1字文字列の集合[訳注1)]

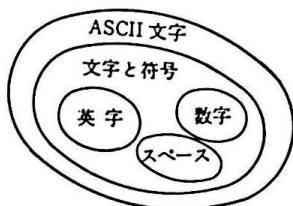
英字　::=　52個の ASCII アルファベット文字からとられた1字文字列の集合[訳注2)]

␣　::=　ASCII 文字のスペース (SP) からなる1字文字列[訳注3)]

構文仕様の中では，ASCII の文字と符号のどの文字を使うこともありうるが，その際はその文字から成る1字文字列を意味する．（このことは構文規則上の用語が構文仕様中に現われるときに常に下線をもつ理由の1つである．）次の印字されない ASCII 文字も構文仕様の中で使われる．スペース (SP または␣で表わされる)，左端復帰 (CR[訳注4)]で表わされる)，改行（LF[訳注5)]で表わされる)，改頁（FF[訳注6)]で表わされる）がそれである．

訳注 1)　ASCII 10進コードで32から126までの文字．

訳注 2)　ASCII 10進コードで65から90までと97から122までの文字．

訳注 3)　ASCII 10進コードで32の文字，従って

訳注 4)　CR:　Carriage Return (ASCII 10進コード 13).

訳注 5)　LF:　Line Feed (ASCII 10進コード 10).

訳注 6)　FF:　Form Feed (ASCII 10進コード 12).

2.5 移送基準

『MUMPS 移送基準』から引用した記述は角括弧で囲ってある.

〔Port: . . . :Port〕

角括弧内の記述は MUMPS 言語についての記述というよりも，応用プログラムを書く人と MUMPS システム言語装備者が応用プログラムの機種間移送性を目標とする場合に，現実的な犠牲を払っても充分受け入れなければならない制限であると解さるべきである.

2.6 例の利用法

この本は読者が正しい理解と解決法を引き出すための正確な基礎を提供しようとしている. 一方用例の部分では形式上完璧でないときには不正確な記述から読者の力によって正しい帰結が得られるように期待している. 従ってこのような例につけられた解説の内容は，ずっと気楽な調子で，しばしばいくつかの例の比較から読者が結論を得るようにしている.

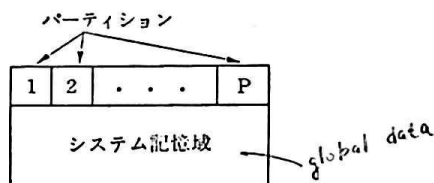
時おり“典型的”という言葉がこの本の文中に現われる. “典型的”という語を含んだ記述は内容を精密に述べたものでなく，一般的な用法を述べたものとして理解してもらいたい.

第 II 部

MUMPS システム モデル

第 3 章
MUMPS システム モデル

Pを正の整数とし，P個のパーティションをもつ MUMPS システムを考えると，これは最大 P 個の MUMPS 処理過程を同時に実行することができる．各“パーティション”は１つの MUMPS 処理過程を実行する環境である．P という値は通常１つの MUMPS システムに固定した（または非常に徐々に変化し得る）性質があるが，必ずしもそのように考える必要はない．

ここで用いるモデルを，P-パーティションをもつ MUMPS システムとすると，そこには (P+1) 個の構成要素がある．すなわちP個のパーティションと１個の「システム記憶域」(System Storage) である．

「システム記憶域」はすべてのパーティションに供給できるいろいろのデータ構造を含んでいる．「システム記憶域」の大部分は伝統的にグローバル データと呼ばれるものから成り立っている．言語の本体とデータとを充分区別して認識してゆくために，我々はここで「システム記憶域」のうち伝統的にグローバル データと呼ばれる部分を「名前付きシステム記憶域」(Named System Storage) という用語で呼ぶ．

「システム記憶域」はその他に１個の時計と更に P 個の部分からなる３種のデータ構造をもつ．３種のデータとは「Lock リスト P-ベクトル」(Lock List

P-vector),「Open リスト P-ベクトル」(Open List P-vector),「作業番号 P-ベクトル」(Job Number P-vector) である．パーティションはそれぞれにこの3種のデータをもつのであるが，これらが「パーティション記憶域」ではなく「システム記憶域」におかれている理由は,「Lock リスト」や「Open リスト」を操作するとき，個々のパーティションが他の (P-1) 個のパーティション用のこれらのリストを知る必要があるからである．同時に「作業番号」にもそれぞれ独特な番号が割り当てられている． システム モデルは個々のパーティションが「システム記憶域」とは直接情報交換ができるが，2つのパーティションが互いに直接情報交換はできないという前提で設計されている．「システム記憶域」の詳細な構造については第4章で述べる．

各パーティション内には次のような構成要素がある．

1. インタプリタ訳注1)
2. 「パーティション記憶域」(Partition Storage)
3. 「パーティション スタック」(Partition Stack)

このモデルでは各パーティションはそれぞれ固有のインタプリタを与えられているが，これは中央処理に対比して分散処理をおこなおうとするためではなく，時分割法に従ってその都度プログラムが場所を取らねばならないことを回避するためである．システム (プログラムに対するシステムという意味で) の設計によって決定される序列依存性を回避する方向で更に段階を進めれば，数個のインタプリタと「システム記憶域」間の対話 (値を読みこんだり，値を書き出したり，下位樹枝訳注2)を消したり，連鎖 (chain) を作ったり，リストを作ったり消したりなど) は互いに独立しているものであり，しかもそれらは同時に起こるものではないと定義される．ここで定義されないものとしては，個々のインタプリタの絶対的な，または他のインタプリタに対する相対的な実行速

訳注 1) Interpreter; ソース言語コードから機械コードへの変換をプログラム実行時に行う通訳部．アセンブラ，コンパイラに対比される．

訳注 2) 樹枝; tree (トリー) とも呼ばれる．

度である.

「パーティション記憶域」はパーティションに局在して Do または XECUTE 命令を実行しても増加しない複数のデータ構造をもっており，これらは「名前付きパーティション記憶域」(Named Partition Storage)（すなわちローカルデータ または“シンボル テーブル”というもの),「略式グローバル指標」(Naked Indicator),「テスト スイッチ $T」(Test Switch),「現用装置指定子」(Current Device Designator) である.「パーティション記憶域」については第5章で詳しく解説する.

「パーティション スタック」は順序を正しく制御するために積み重ねて（スタック）おかねばならないすべての記憶を入れている．このシステム モデルでは行ポインタ，行バッファ，文字指示子のような状態情報と一緒にルーチン体もここに記憶されている．このモデルでは式の評価を行っているときの詳細な積み重ね記憶処理（スタッキング）には関係せず，そのため値スタックというようなものをもっていない．第6章で「パーティション スタック」とその実行順序との関係を詳しく解説する.

第 4 章
システム記憶域

P個のパーティションからなるシステムの「システム記憶域」は次のような要素から成っている.

1. 「名前付きシステム記憶域」のデータ構造
2. 次に示すような3つのP–ベクトルがあってそれぞれのP–ベクトルの要素はパーティションとの間に1対1の対応をもつ.
   a. 「作業番号P–ベクトル」. 各要素は整数である.
   b. 「Lock リスト P–ベクトル」. 各要素は「名前付き システム記憶域」の節指定子, あるいは「名前付きパーティション記憶域」の節指定子の並記である.
   c. 「Open リスト P–ベクトル」. 各要素は「装置指定子」の並記である.
3. 「時計レジスタ」(Clock Register)

4.1 名前付きシステム記憶域

「名前付きシステム記憶域」は有限の樹枝構造であり, その節の各々の上に, いわゆる"属性ブロック"を記憶しているレジスタがついている. その内容は以下に述べる一種の制限に従っている. 樹枝は階層に分かれ階層nにある節はすべて根節("登録節")からn層離れた階層の中にある.

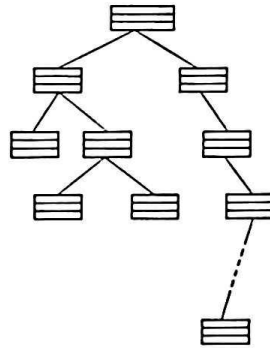
登録レベル（階層０）

無添字レベル（階層１）

単一添字レベル（階層２）

二重添字レベル（階層３）

⋮

ｎ重添字レベル（階層 n+1）

上の図で長方形で示した各属性ブロックは「名前」(Name),「D」(D),「値」(Value) という属性の入っている３部分の記憶レジスターからなる．

名前:	^INV¢3¢5¢
D:	1
値:	2×4×8 : 160

<u>各節の属性ブロック内容に関する法則</u>

1. 「D」属性に入り得るのは４種のデータ列である．

   ０

   １

   １０

   １１

2. 「D」が０または１０であるとき「値」属性は“定義されていない”と言われ，属性ブロックの「値」入力はないと言われる．
   「D」が１または１１であるとき「値」属性は“定義されている”と言われ，属性ブロックの「値」入力は，空文字列を含むいかなる文字列でもあり得る．

3. 階層 k（k≠0）での「名前」属性はｋ重になっている．

N¢S1¢S2¢. . . ¢Sk-1

a．ここで分離子¢はデータとして用い得る文字集合以外の独特の分離子で，印刷の便宜上選ばれたに過ぎない．

b．k重の各要素はデータ文字列である．

c．〔Port: 各Siデータ文字列のつづりは負でない整数の構文規則を満足する（12.4参照）: Port〕

（注：この移送基準の記述法については，2.5参照のこと）

d．Nで表わすデータ文字列のつづりは^名前の構文規則（12.4参照）を満足する．

節や属性ブロックはデータ構造の部分であると理解されるが，その分枝構造[訳注1)]はデータではない．つまり分枝構造は上下関係を示すために描かれているに過ぎないことに留意すべきである（以下参照）．

属性ブロック内容と樹枝構造間の連結に関する規則

1. 各節は属性ブロックをもつ．
2. 登録レベルで節（“登録節”）の属性ブロックは入力を「D」ただ1つしかもたず，その内容もØか1Øに限られていることに注意すべきである．
3. 樹枝の最初の状態（初回の処理が始まる前）は，そこには単に登録節だけがあり，D=Ø である．
4. 以下の定義の中でN(i)を階層iのいかなる節であってもよいものとする．

　a．もしN(k)とN(k+1)が分枝によって連絡されておれば，N(k+1)はN(k)の“直接下位”であると言い，N(k)はN(k+1)の“直接上位”であると言う．

　b．もし j<k であり節の序列がN(j), N(j+1), . . ., N(k)に限られる

訳注 1)　branch

とき，〔j, k−1〕の中のどの i をとっても，N(i) が N(i+1) の直接上位であるとき，N(j) は N(k) の“上位”であり，N(k) は N(j) の“下位”であると言う．

規則：　すべての正数 j について N(j) が N(j+1) の直接上位にあると言えるのは，N(j+1) の「名前」属性の内容から ¢S<u>j</u> を削除したときに，N(j) の「名前」属性の内容が得られたときに限る．

5. 節 N(k) で D=Ø または 1 であるならば，N(k) は下位をもたない．その逆は必ずしも真でなく，D=1Ø または 11 をもつような N(k) が“終末節”（下位をもたない）として存在するかも知れない．ただしこのような状態は N(k) が直接下位をただ 1 つしかもたないとき，その直接下位の名前に対して選択的 KILL を実行させた直接結果としてのみ生じるのである．

4.2 Lock リスト P-ベクトル

システムのパーティションは，1 から P まで番号を付けられ，それぞれ 1 個の“Lock リスト”と対応している．これら P 個の「Lock リスト」が集合して“Lock リスト P-ベクトル”になり，これは「システム記憶域」にある．各「Lock リスト」には何も入っていないこともあり，また<u>節指定子</u>の構文規則を満足するデータ文字列のリストが入っていることもある．それらはデータ文字集合以外の特別の分離子，ここでは ¢¢ で表わす，で分離されたものである．

<u>節指定子</u>は「名前付きシステム記憶域」か「名前付きパーティション記憶域」のどちらかの階層 k の節の属性ブロックの「名前」の部分に生じる k 重（k≠Ø）のデータを表わしその構造は次の通りである．

N¢S<u>1</u>¢S<u>2</u> . . . ¢S<u>k−1</u>

a．ここで分離子 ¢ は ASCII データとなりうるアルファベット以外のものである．

b．k 重の個々の内容はデータ文字列である．

c．〔Port: 各S_i_データ文字列のつづりは負でない整数の構文規則を満足する（12.4参照）: Port〕.

d．Nというデータ文字列のつづりは名前あるいは^名前の構文規則（12.4 参照）を満足する.

Lock リスト 1:〔節指定子〔¢¢節指定子〕...〕

⋮

Lock リスト P:〔節指定子〔¢¢節指定子〕...〕

下位排他規則（Descendant Exclusivity Rule）

「Lock リスト」iのいかなる節指定子も i≒j とするとき,「Lock リスト」jの指定子とは, 第一義的な序列関係をもたない.（解釈：1つのパーティションの「Lock リスト」であげられた節は, 他のいかなるパーティションの「Lock リスト」であげられた節の, 上位節あるいは下位節でもない.）

「Lock リスト」の中に節指定子があるということと,「名前付きシステム記憶域」または「名前付きパーティション記憶域」の中に節が存在するということは, 全く無関係のことである.

初期にはすべて「Lock リスト」は空である. 詳しく言えば, パーティションiの処理過程が始められるとき,「Lock リスト」iは空である.「Lock リスト」の内容は次に示すうちの1つの活動の直接結果としてのみ変化する.

1. パーティションiの過程の終了（HALT の明確な, あるいは暗黙の実行[訳注1)]）で「Lock リスト」iは空になる.
2. パーティションiの引数をもたない LOCK 命令の実行で,「Lock リスト」は空になる.
3. パーティションiの引数をもつ LOCK 命令の実行：

a．初めに「Lock リスト」iを空にする.

訳注 1)　HALT の明確な実行とは, プログラムに HALT 命令があって, それを実行すること. HALT の暗黙の実行とは QUIT または eor が実行され, かつスタックが空の場合.

b．次に引数に示されたすべての節指定子のリストを設定する．ただしこれは「下位排他規則」がこれによって違反されない場合に限られる．違反されれば，「Lock リスト」は空のままである．

4.3 Open リスト P-ベクトル

“装置指定子”の構文規則は，各言語装備者によって指定される．しかし次のことだけは，ここで述べておくことができる．

1. 文字列データであること．
2. 順次文字を出力し，そして／または順次文字をとりこむ“装置”の特定の1個を指す．

システムのパーティションは1からPまで番号を付けられ，それぞれ1個の“Open リスト”と対応している．これらP個の「Open リスト」が集合して“Open リスト P-ベクトル”になり，これは「システム記憶域」にある．各「Open リスト」には何も入っていないこともあり，または非データ分離記号¢¢によって分離された装置指定子のリストが入っていることもある．

「Open リスト」1:〔装置指定子〔¢¢装置指定子〕...〕

⋮

「Open リスト」P:〔装置指定子〔¢¢装置指定子〕...〕

装置排他規則

「Open リスト」iの装置指定子は，i≠jである場合，「Open リスト」jにある装置指定子と同一の装置を指定することができない．(解釈：パーティションは装置を占有する．)

最初すべての「Open リスト」は空である．詳しく言えば，パーティションiの過程が始められるとき，「Open リスト」iは空である．(例外については5.4の「主装置慣例」の解説を参照のこと．)「Open リスト」の内容が変化す

るのは，次に示すうちの１つの活動の直接結果としてだけである．

1. パーティションｉで OPEN を実行すると，引数並記が定義するこれらの装置指定子が「Open リスト」ｉに付け加わる．
2. パーティションｉで CLOSE を実行すると引数並記が定義するこれらの装置指定子が「Open リスト」ｉから取り除かれる．（CLOSE に伴う副次的効果の可能性については，5.4の「主装置慣例」の解説を参照のこと．）
3. パーティションｉの過程の終了（HALT の明確な，あるいは暗黙の実行）は「Open リスト」ｉを空にする．

4.4　作業番号 P-ベクトル

処理過程が実行されるためにはそれが“活動状態”になければならない．過程を活動状態にする手段は標準 MUMPS では定義されない．一たん過程が活動状態となると，それは HALT あるいはそれに相当するものが実行されるまで活動状態のままである．（6.2の2．を参照のこと．）

活動状態にある過程をもつシステムのパーティションは，それぞれ独自な正の整数である“作業番号”と対応する．活動していないパーティションは「作業番号」として0をもつ．これらＰ個の「作業番号」の集合は「システム記憶域」内で“作業番号 P-ベクトル”を作る．１つの過程の「作業番号」はそれが活動状態になったとき割り当てられ，それは過程が活動状態である間固定されている．その値は式子として特殊変数 $Jを実行させたときに直接取り出すことができる．活動状態にある２つのパーティションが，同時に同一の「作業番号」値をとることはない．ある種の言語装備においては，活動状態にあるパーティションの「作業番号」が単にパーティション番号である場合があるが，これは必ずしもすべての場合にはあてはまらない．

4.5 時　　　計

この記憶レジスタの内容は常に定義されており，式子として特殊変数 $H を実行すると直接取り出すことができる．時計値は，正確な日付と時間を表わし，コンマによって分けられた2個の整数 D, S の形をとる．D は日付のカウンタであり，S は D の各値に対して Ø から 86399 まで作動する秒のカウンタである．Dの値は各日の真夜中から次の真夜中まで変わらない．真夜中の瞬間に，Dは1だけ数を増し，SはØにセットされその後真夜中からの秒数を入れる．基準となる時計値Ø, Øは1840年12月31日の初めの1秒間である．

第 5 章

パーティション記憶域

システムの「パーティション記憶域」のP個のパーティションは，それぞれ次の要素をもっている．

1. 「名前付きパーティション記憶域」のデータ構造，通常“ローカル データ”または“シンボル表”というもの．
2. 「略式グローバル指標」
3. 「テスト スイッチ」
4. 「現用装置指定子」
5. 「現用装置水平カーソル」および「現用装置垂直カーソル」

5.1 名前付きパーティション記憶域 (Named Partition Storage)

各パーティション内の「名前付きパーティション記憶域」は次のものを除いて「名前付きシステム記憶域」(4.1参照) と同一の規則で作られたデータ構造である．

各節の属性ブロックの内容に関する規則を次のように読み替える．

「名前」属性のNで表わす部分のつづりは，^名前の構文規則の代わりに，名前の構文規則を満足する．

属性ブロックの内容と樹枝構造間の連結に関する規則3を次のように読み替える．

各パーティションには，それ自身の独立した樹枝があり，パーティションiでの処理の開始の際，樹枝には登録節のみがあり D=Ø である．

5.2 略式グローバル指標 (Naked Indicator)

「略式グローバル指標」の内容は空（“未定義”）か，または節指定子（4.2参照）の構文規則を満たすものかのいずれかである．

「略式グローバル指標」はパーティション内で次の動作が行われたときにのみその直接結果として変化する．

1. 過程が開始するとき「略式グローバル指標」の内容は未定義となる．
2. 添字をもたないグローバル変数の実行は「略式グローバル指標」の内容を未定義とする．
3. グローバル変数ないし略式グローバル参照で，それの直接上位節が存在しないか，D=0 または1であるものを実行したときには，「略式グローバル指標」の内容は未定義となる．しかしグローバル変数ないし略式グローバル参照が SET 命令の引数の中で境界子＝の左側にくる場合にはこれは起こらない．
4. 上の場合を除いて，グローバル変数ないし略式グローバル参照を実行すると，直接上位の「名前」属性の内容が「略式グローバル指標」の中に書き込まれる．すなわち階層kの節指定子の中から，最後の¢Sk-1を取り除いた残りが，「略式グローバル指標」の中に入れられることになる．

「略式グローバル指標」が未定義のときに，略式グローバル参照を実行するとエラーとなる．「略式グローバル指標」が定義されていれば，^(式1，...，式i）という略式グローバル参照でS1＝式1の値，...，Si＝式iの値，であるとすればこの参照を実行することは，節指定子が

「略式グローバル指標」の内容 ¢S1¢ ... ¢Si

となり，これを利用したあとで，¢Siが除かれた値が「略式グローバル指標」の中に入るということである．

5.3 テスト スイッチ（Test Switch）

「テスト スイッチ」レジスタの内容は，次の3個のデータ文字列のうちのどれか1つである．すなわち，空（“未定義”），0（“偽”），1（“真”）である．内

容が定義されていれば，式子として特殊変数＄Tを実行することによって，直接取り出すことができる.

「テスト スイッチ」の内容は，パーティション内で次の動作が行われたときにのみその直接結果として変化する.

1. 引数をもつ IF の実行は，引数の 真偽値解釈 の 結果を「テスト スイッチ」に入れる.
2. 引数に時間制限をつけた LOCK, OPEN, または READ はその実行時，ある条件をためすことになる. その条件とはそれぞれ，「下位排他規則」に違反せずパーティションの「Lock リスト」を設置できるか，「装置占有規則」に違反せずにパーティションの「Open リスト」を設置できるか，または時間制限の後，実行を再開させる前に入力データの文字列が明確に終わったかという条件である. もし条件が満足されれば「テスト スイッチ」は値1をとり，満足されないなら値∅をとる.

＄T の実行で「テスト スイッチ」の内容は 直接取り出せるがこのほかに，過程は ELSE または IF を実行させて，それらを 含むラインの残りの実行を「テスト スイッチ」の値に従って条件づけることができる.

5.4 現用装置指定子 (Current Device Designator)

“装置” と “装置指定子” の定義については，4.3の 第1文節を参照のこと.「現用装置指定子」レジスタの内容は，空 (“未 定 義”) であるか，さもなくば装置指定子の構文規則を満足させる データ文字列 である.「現用装置指定子」の内容は，式子として特殊変数＄Iを 実行することによって直接取り出すことができる.

“現用装置” は READ または WRITE を実行するとき，データが伝送されるようにパーティション内でただ1つ指定されている装置である. 現用装置の指定が定義されていないとき (すなわち「現用装置指定子」の内容が空のとき)，READ または WRITE の実行の結果は定義されていない. 定義された指定が

ある場合に「現用装置指定子」の指定する装置が現用装置となる．

現用装置占有規則 (Current Ownership Rule)

パーティションｉの現用装置は常に「Open リスト」の中に指定されていなければならない．(解釈：処理過程は，あらかじめ装置を占有していなければ，その装置を現用装置として宣言することができない．また処理過程が現用装置を CLOSE すると，装置指定子は「現用装置指定子」からも「Open リスト」からも除かれる．)

言語装備者は「現用装置指定子」にあらかじめ入れておく内容と，現用装置を指定する CLOSE に対して起こる反応の内容について，選択権がある．その１つの例として“主装置慣例” (Principal Device Convention) があり，それを次に示す．

1. １つの処理過程に対して“主装置”と呼ばれる，１つの装置Ｄというものがある．この主装置の定義は，言語装備者が与え，処理過程の続くかぎり変わらない．
2. パーティションｉの処理過程の開始の際「Open リスト」ｉは，Ｄ（それのみ）を指定する．そしてパーティションｉの「現用装置指定子」はＤを指定する．
3. 「現用装置指定子」を空にするような CLOSE が実行されるときは，いつもＤという指定子が「現用装置指定子」と「Open リスト」の中に入ってくる．(ただし，これはＤがもうそれらに入っていない場合で，かつこの操作が「現用装置占有規則」に違反しない場合に限ってである．)

もし言語装備者がこの処理過程に関して「主装置慣例」に従わないならば，「現用装置指定子」は初めは空であろう．そして現用装置を指示する CLOSE の実行の結果もこれは空となる．

上に述べられた特別の場合を除いて「現用装置指定子」の内容は USE の実

行の直接結果としてのみ変化する.

USE が実行されるとその引数が示す装置指定子は「現用装置指定子」の内容にとって変わる. ただしこの場合引数が示す装置指定子は「Open リスト」も指定している装置を指定していなければならない.

5.5 現用装置水平カーソルおよび垂直カーソル (Current Device Horizontal and Vertical Cursors)

「システム記憶域」の中で, 各装置の状態の情報として蓄えられているものに, 1つの装置当たり2つのレジスタすなわち X, Y がある. それぞれを水平カーソル[訳注1], 垂直カーソルと言う.「パーティション記憶域」には2つのレジスタすなわち CX, CY があり, それぞれを「現用装置水平カーソル」,「現用装置垂直カーソル」と言う. これらは現用装置に関する X と Y の値を提供する. CX と CY の内容は, 式子として特殊変数 $X, $Y を実行することに直接取り出すことができる.

「現用装置指定子」の内容が変化するとき, CX と CY の内容は, それまでの現用装置の X, Y レジスタに置かれ, 新しい現用装置の X, Y レジスタが, CX, CY レジスタに置かれる. このように装置占有状態が変化しても, 装置のよって, カーソルの状態は失われない.

各カーソルの内容は, 構文規則として<u>負でない整数</u>を満足するデータ文字列である. 各カーソルの最初の値は, その装置を現用装置としている何らかの処理過程が#書式を実行したときに定義される.

現用装置を変えるときの CX, CY の変化以外では, CX, CY は READ と WRITE の実行中になされる書式操作と文字の移動によってのみ変化する. 各書式操作は, 次のように CX と CY の値を制御する.

訳注 1)　カーソル；Cursor, 文字が次に入/出力される位置を示すマーク.

!　CX を∅とし，CY に1を加える．

#　CX, CY を∅にする．

?n　CX の中に max (n, CX) の値が入る．

READ と WRITE の実行の間になされるデータ移送の結果としての CX, CY への効果は，次のようなことを目標として MUMPS 言語標準で規定されている．しかし標準のわく組を計画した委員たちは，ある種の装置ではオペレーティング システムの制約のために，すべての目標を達成することはできないことを認識している．これらの目標は以下に記述されており，実行可能な範囲でできるだけ 100 %近く達成させることを言語装備者に期待している訳注1)．

1. 文字を移送することが CX, CY に及ぼす効果は，その文字が移送された都度記憶されるべきこと．
2. 文字を移送することによる CX, CY への効果は，各々の文字の位置認識を保つための機能であって，その文字の内容とか，READ または WRITE のいずれが実行されているかということとは無関係であるべきこと．
3. CX, CY への次に示す効果が規定されるべきこと．

文字と符号1文字に対して：CX に1を加える．
バックスペース (BS)：CX=max(∅, CX−1) とする．
改行 (LF)：CY に1を加える．
左端復帰 (CR)：CX=∅ とする．
改頁 (FF)：CX=∅, CY=∅ とする訳注2)．

訳注 1)　既存のオペレーティング システムに装備される場合など100%の達成ができないことがある．この場合はユーザー マニュアルに理由と共に明記しておくことを MDC MUMPS 検定会議では勧告している．

訳注 2)　これら以外の制御文字，例えば HT (Horizontal Tab), VT (Vertical Tab) などの CX, CY への効果が規定されている言語装備もある．

第 6 章

パーティション スタックと実行順序の制御

「パーティション スタック（積み重ね記憶）」(Partition Stack) は MUMPS 処理過程の実行順序を制御するための，最後に加わったものが最初に除かれるデータ構造である．

スタックの各項目は次の5つの成分をもつ．

1. ルーチン体
2. ルーチン体の1つの行を示す「行ポインタ」．これはそのルーチンの終了点にある eor[訳注1)]を指定することもできる．
3. 「行ポインタ」によって示される行の（おそらく修飾された形での）複写をもつ「行バッファ」
4. 「行バッファ」の1つの文字を指示する「文字ポインタ」
5. 「For 範囲スイッチ」と呼ばれる，2つの値をもつ指標

〔Port: 移送基準に従って作るルーチンは「パーティション スタック」が15重を超える入れ子を作ってはならない :Port〕

6.1 通常の実行順序

MUMPS 処理過程の実行は単元（アトム）操作の連続であり，1つの単元操作とは1文字を実行することである．現在個々の文字が実行されつつあるルーチン体は常にスタックの最も上の部分にある．このルーチン体の中では，実行される文字の順序が，以下で説明する規則に従って記述されている．通常次の規則が適用される．

訳注 1) eor; end of routine（第7章参照）．

1. “通常の実行順序”は HALT, IF, ELSE, FOR, QUIT, eor の実行，または DO, GOTO, IF, もしくは XECUTE の引数の実行，または間接指定によって変更される場合を除き常に優先される．規則2と3で通常の実行順序を述べる．
2. 1つの行の実行は，その行の複写が「行バッファ」にある間に行われる文字実行の順序として定義される．新しい行を示すために「行ポインタ」が変化すると次のことが起こる．
   a．1s のすぐ右の文字から始まって，eol[訳注1)]をも含んだ新しい行の複写が，ルーチン体からもってこられ「行バッファ」内におかれる．
   b．「文字ポインタ」が「行バッファ」の最も左の文字を指示するように設定される．
3. 「文字ポインタ」によって指示される1文字の複写が得られ，それが実行される．もし文字が eol であって「For 範囲」スイッチがオフであれば，その効果は「行ポインタ」を次の行（または eor）まで進めることになる．もし文字が eol でなければ，文字実行前に「文字ポインタ」が右側の次の文字に進められ，そして前の文字が実行される．

従って通常の実行順序はルーチン体内の行内の文字の完全な左から右への連続した実行である．新しいルーチン体が「パーティション スタック」の最上部に入ったとき最初に実行されるべき行は，そのルーチンを指示する DO の引数内で明確に認識される．これがない場合，すなわち DO の引数が行指定をもたないか，または行指定なしに処理を始めるような場合は，ルーチン体の最初の行を意味する．

処理過程の初期状態すなわち最初の文字の実行の直前では「パーティション スタック」は1つのルーチン体を含み，そして「行ポインタ」と「文字ポインタ」とが最初に実行される行の最初の文字を指し，「For 範囲」スイッチはオフに設定される．

訳注 1) eol; end of line（第7章参照）

6.2 順序の変更

通常の実行順序の変更は，次の方法で起こる.

1. HALT の実行は処理過程を終わらせる. 処理過程の終了のとき，その処理過程の入っているパーティションkに関連した「システム記憶域」内では，次の動作が起こる.
   a. 「作業番号 P-ベクトル」の k 番目の要素は0となる.
   b. 「Lock リスト P-ベクトル」の k 番目の要素は空となる.
   c. 「Open リスト P-ベクトル」の k 番目の要素は空となる.
2. 「For 範囲」スイッチがオフであって QUIT を実行する場合，または eor を実行するときは「パーティション スタック」の最上部の要素がとれてしまう. もしそれによって「パーティション スタック」が空になるなら，処理過程はあたかも HALT が実行されたかのように終了する. もし「パーティション スタック」のルーチン体が，なお存在するならば，次に実行される文字は，新しいスタックの最上部の「文字ポインタ」が指示する文字である.
3. 次に述べられている条件のうちのどれかを実行すると，「文字ポインタ」は右方に走査し，各文字をとってくるが，「行バッファ」の eol を指示するまでは実行をしない. すなわち次の場合の実行の効果は，それの右側にあるラインの残りの実行を抑制することである.
   a. 「テスト スイッチ」が0であるような引数なしの IF の実行.
   b. 「テスト スイッチ」が1であるような ELSE の実行.
   c. その値（真偽値解釈をして後）が0である IF の引数の実行.
4. DO 引数の実行は，新しい要素を「パーティション スタック」に押し込むことになる. その手順を以下に解説する.
   a. もし DO 引数がルーチン名を含んでいるならば
      (1) ルーチン体はルーチン名によって指定されるものである.
      (2) 「行ポインタ」は，DO 引数の指定する行を指示するように設定さ

れる．すなわち，もしDo引数が行参照をもたない場合は「行ポインタ」はルーチン体の最初の行を指す．

(3) 「行バッファ」が設置され，「文字ポインタ」はその1番左の文字を指している．

(4) 「For 範囲」スイッチはオフとなる．

b．もしDo引数がルーチン名を含んでいないならば

(1) ルーチン体はスタックの最上部にあるものの複写である．

(2) 「行ポインタ」はDo 引数の指示する行を指す．その後上記の(3)，(4)が実行される．

5. GOTO 引数の実行は次の2つの動作を続いて起こさせる．

a．「For 範囲」スイッチがオフであってQUIT が実行されたかのように，スタックの最上部の要素は消えてしまう．

b．GOTO の引数と同じつづりのDo の引数が実行されたかのように，スタックの最上部の要素が積み重なる．

6. XECUTE 引数の実行は，以下の修正を行ったDo 引数の実行によく似ている．

a．ルーチン体は，スタック最上部にあるものの複写である．

b．「行ポインタ」は eor のすぐ前の，そのルーチン体の最後の行を指す．

c．「行バッファ」に次の文字列が設定される．

XECUTE 引数の値 eol

そして「文字ポインタ」はその最も左の文字に設定される．

7. 命令語 FOR の実行は，その前の状態にかかわらず「For 範囲」スイッチをオンにする．最初の for パラメータ内のすべての式を評価した後に，「パラメータ マーカ」(Parameter Marker) と呼ばれる独特な標識文字が，「行バッファ」の今評価された for パラメータとそのすぐ右の分離子との間に入られる．「文字ポインタ」はその後，for パラメータ並記

に続くスペースの右側の最初の文字へと右に動く.（もし for パラメータ並記にすぐ eol が続くなら,「文字ポインタ」は, この eol を指す.）「範囲マーカ」(Scope Marker) と呼ばれるもう1つの独特な文字が, この文字のすぐ左に入れられ, そして通常の実行が再び始まる. 通常の順序内で次のような中断が続いて起こり得る.

a. もし「For 範囲」スイッチがオンであるまま eol が実行され, そしてその範囲が現在の for パラメータのもとで再び実行されねばならないときは,「文字ポインタ」は最も右の「範囲マーカ」の右側の文字に再び位置を定め, そして通常の実行順序が再開する.

b. もし「For 範囲」スイッチがオンであるまま eol が実行され, この範囲が現在の for パラメータのもとで再び実行されないなら,「文字ポインタ」は「パラメータ マーカ」をさがすために左へ走査する.

(1) もし「パラメータ マーカ」が見つかり, そのすぐ右の文字がスペースまたは eol なら（すなわち, 最後の for パラメータの制御下にあるその FOR の範囲がちょうど実行されるなら）「パラメータ マーカ」とその右の「範囲マーカ」は消され, そして上の b. で述べられた方向への走査が再び始められる.

(2) もし「パラメータ マーカ」が見つからないときは,「For 範囲」スイッチはオフになり, 通常の eol の実行（6.1の第3分節に述べられている）が起こる.

(3) もし「パラメータ マーカ」が見つかり, その右の文字がコンマなら「パラメータ マーカ」は消され, その右の for パラメータが評価され,「パラメータ マーカ」がこの for パラメータのすぐ右に入り,「文字ポインタ」は「範囲マーカ」の右の文字を指し示すために動き, そして通常の実行が再開される.

c. 「For 範囲」スイッチがオンである際の QUIT の実行は,「文字ポインタ」をそれが「パラメータ マーカ」に出会うまで左に走査させる. この「パラメータ マーカ」とその右の「範囲マーカ」は消され

る．その後上記の7a．から始まるeolの手続きが実行される．

d．「For 範囲」スイッチがオンである際のGOTOの実行は，5. に述べられたことと同じである．

8. @（間接指定の符号）の実行は，次のステップを起こさせる．

a．@の右の式子が評価されるにつれ「文字ポインタ」は右に動く．この評価の間に再度@に出会い得ることに注意すべきで，そのときは8. の全部が繰り返される．

b．式子が評価された後，式子の値は式子のすぐ右側で「行バッファ」に入れられ，「文字ポインタ」はこの値の最も左の文字を指し示す．（もしこの値が空なら「文字ポインタ」は，式子の右の最初の文字を指し示し，以下のステップの効果はなくなる．）

c．式子の評価から生じた「行バッファ」内の各々の文字は，「文字ポインタ」によって指定され，それの複写が取り出されるので，その文字は「行バッファ」から消される．効果は間接指定が起こったという痕跡を残さないので，1つの間接指定の用例が1つのFORの範囲の中で，何回呼ばれてもよい．

第 III 部

言　　語

第 7 章
ルーチンと行

ルーチンはパーティションにおいて互いに交代するプログラムの単位である．プログラムとパッケージは順序づけられていないルーチンの集まりである．ルーチンを実行のために呼び出すにはそのルーチン名を指定する．ルーチン名の従うべき構文規則は，名前と同一のものである．

<table>
<tr><td rowspan="2">ルーチン名 ::=</td><td>%</td><td>〔数字</td><td rowspan="2">…</td></tr>
<tr><td>アルファ</td><td>アルファ〕</td></tr>
</table>

〔Port: 9個あるいはそれ以上の文字を含むルーチン名はそれらの左の8文字のみを基準として識別され区別される．小文字のアルファベットはルーチン名の中で使われてはならない． :Port〕

ルーチンの構文について考えるとき，我々はルーチンの交代のアルゴリズム上，それらのルーチンを，外部記憶装置に長くひも状に蓄えられた文字列として考えている．この意味において1つのルーチンは2つの部分から成る．すなわち交代のときに識別するためのルーチン名を含む頭部と，一連の行から成り実行されるコードを含む体部とで作られる．

ルーチン ::= ルーチン頭 ルーチン体

ルーチン頭 ::= ルーチン名 eol

ルーチン体 ::= 行〔行〕 … eor

交代するとき3つの分離子 1s，eol，および eor は次の ASCII 文字，あるいは制御文字の組を意味する．

1s ::= SP

eol ::= CR LF

eor ::= CR FF

従って，もしルーチンが ASCII 制御慣例に従っている装置に印刷されるならば，ルーチン名は最初の行に印刷され，実行されるコードは残りの行に印刷されるはずである．

ルーチンの構造と同様に各行は2つの部分から成る．行頭は行を識別するためのラベルを任意に含んでもよい．そして行体があればその中に実行可能な命令を含んでいる．

行 ::= 行頭 行体

行頭 ::= 〔ラベル〕 1s

行体 ::= ⎧命令〔␣命令〕...〔␣注釈文〕⎫ eol
　　　　 ⎩　　　　　注釈文　　　　　⎭

ラベル ::= | 名前 |
　　　　　 | 整数字列 |

注釈文 ::= ;〔文字と符号〕 ...

〔Port: 1行の文字の総数は行頭全体と，eol を除外した行体とを合わせて255字を越えてはならない. :Port〕

ラベルは行を識別するのに使われる．行参照は DO と GOTO 命令および $TEXT 関数で行われる．DO と GOTO はどのルーチン内の行を参照することもできるが，$TEXT は「パーティション スタック」の最上部のルーチン体だけを参照する．

1つの与えられたルーチン内で，行頭がラベルを含んでいる行はそのラベル名を示すことによって指定できる．このように行参照の最も簡単な形はラベルの起用である．もっと一般的に言えば，与えられたルーチンのいかなる行でも

その行より前にあって行頭にラベルを含んでいる行を参照することによって指定し得るのである．その形式ラベル＋整数式（この整数式の整数評価値を i とする）は，“行頭にラベルを含んでいる行の後に来る i 番目の行”を意味している．参照ラベル＋0はラベルと同義である．整数式に負の値は許されない．

どんなルーチン内でも1つのラベルのつづりは，行頭に一度より多く現われてはならない．ラベルが全部数字から成るときは，頭に0を付けてもつづりとしては意味がある．つまりラベル01はラベル1とは違う．〔Port: 9個あるいはそれ以上の文字を含むラベルは，左の8文字のみをもとにして識別され区別される．:Port〕

一般的形式の行参照では，間接指定が指定するのは，ラベル部分にのみゆるされ，行参照全体に適用してはいけない訳注1)．

行参照 ::= dラベル〔＋整数式〕

dラベル ::= | ラベル / @式子 V dラベル |

＄TEXT 関数はあるルーチンの終点を越えて行を参照する場合や，そのルーチン内の行頭に見当たらないラベルにもおよぶ行参照を処理する型破りの定義をもっている．DO と GOTO はこのような型破り定義はできないので，これらの命令では以下の参照の条件は許されない．

1. 指示されたルーチンの行頭に現われないラベルのつづり．
2. ルーチン体内では行を見つけられない程大き過ぎる整数式の値．

DO と GOTO は行と同様にルーチンを指定することができる．そのためこれらの引数は入口参照と言われるもっと一般的な形をしている．入口参照は構

訳注 1) この制限は＄TEXT 関数にあてはまる．DO, GOTO 命令の引数がルーチン参照をもたない場合はこの限りでない（8.2.3, 8.2.6 および 10.2.12 参照）．

文規則上3種の場合に分類区別できる.

1. 入口参照は行参照の形式をもつ. その場合 ルーチン体は ルーチン スタックの最上部のルーチン体である.
2. 入口参照は^ルーチン参照の形式をもつ. その 場合ルーチン参照で表示されたルーチンの第1行目を意味する.
3. 入口参照は行参照^ルーチン参照の形式をもつ. その場合 指定されたルーチンの指定された行を意味する.

入口参照 ::= | 行参照〔^ルーチン参照〕 | ^ルーチン参照 |

ルーチン参照 ::= | ルーチン名 | @式子 V ルーチン参照 |

第 8 章
命　　　令

8.1 予備的定義

8.1.1 命令の構文規則

第7章において構文規則の対象としての命令が行体内でどのように配置されるかを示した．この章では命令自体の構文規則と意味論を明確にする．

行体の定義（第7章）の中で命令の配置されている場所に8.2で定義されるどの命令を代入してもよい．すべての命令は，SET のようにすべて大文字の記憶符号で書かれた命令語で始まる．

命令は基本的に2つの範疇に分かれる．ある命令語はどちらの範疇に属することもできる．

1. 引数を持たない命令　　引数を持たない命令の構文規則は，命令語と多くの場合1個のスペースを付けたものである．
   a．もし引数を持たない命令語が eol の直前にあれば，命令語の後にはスペースはない．

   QUIT eol

   b．もし引数を持たない命令が eol の直前にない場合には，命令語の後には正確に2つのスペースが付く．第1のスペースは命令の一部分であると考えられ，第2のスペースは，この命令と次の命令ないし注釈文との間隔であると考えられる（第7章の行体の構文規則参照）．

2. <u>引数を持つ命令</u>　引数を持つ命令の構文規則は，命令語に1個のスペースをつけて，それに引数ないし引数並記を従えたものとなる．

IF␣X=Y␣DO␣A␣QUIT : X=1␣␣DO␣B <u>eol</u>

8.1.2 命令の定義の形式

8.2の各命令の定義は次のような形式に統一して行う．

1. <u>構文規則</u>　命令の最も示唆に富み，通常最も簡単な構文がここに示される．
2. <u>詳解</u>　1.で与えられた構文の形の変化を論じる．これらの詳細は次の5つの項目に分けて8.1.3で説明する．
   a．命令語の短縮
   b．命令語の後付け条件
   c．多引数の並記
   d．引数の後付け条件
   e．引数の間接指定
3. <u>実行</u>　命令の実行の効果が述べられる．
4. <u>前後参照</u>　この命令の定義をはっきりさせるような，関連した記述のあるこの本の他の部分の参照．
5. <u>例</u>　命令の使用例を，説明をつけ，また読者が命令についての有用な推論を行えるような順序で示す．

8.1.3 詳　　解

8.1.3.1 命令語の短縮

命令語が正しく書かれるならば，2つの形のいずれかであり，それは命令の定義にある構文を完全につづったものか，第1文字のみで書かれるかである．他の形は許されない．例えば GOTO␣A の場合

GOTO␣A

G␣A

は許されるが，他の場合，例えば次のようなものは許されない.

GO␣A

GOT␣A

GOING␣A

GOTOHERE␣A

8.1.3.2 命令語の後付け条件

後付け条件とは，コロンとそれに続く真偽値式である．これが組となってある種の命令語または命令引数の任意の接尾語として書かれる.

後付け条件 ::= : 真偽値式

ELSE, FOR および IF を除き，標準 MUMPS によって定義されるすべての命令語は，完全なつづりで，また短縮形で，後付け条件をとることができる.

例えば

SET␣X=1

に後付け条件をほどこせば

SET : Y=2␣X=1

または

S : Y=2␣X=1

(“=” の２つの異なった使用法に注意.)訳注1)

命令語に付けられた後付け条件の解釈は次のようになる.

1. もし後付け条件がなければ，この命令全体が無条件で実行され，実行を禁ずるには ELSE や IF のような外部要素が必要となる.
2. 後付け条件があるときは，その右にあるものが何であれ，実行される前に条件が評価される．条件の真偽値判断が 1 (真) なら，命令全体が実行される．後付け条件の真偽値判断が Ø (偽) ならば，命令のどの部分

訳注 1) Y＝2の＝は関係演算子 (relational operator; 11.2.4 参照).
X＝1の＝は値割り付け境界子 (value assignment delimiter; MUMPS Language Standard I-3参照).

も実行されず，引数も評価されない.

後付け条件の評価は，たとえそれが親命令の実行を禁じても，2次的効果をもつことができる．例えば

SET : $D(^(1,2))=3␣X=X+1

の場合は，Xを決して変えないが，常に「略式グローバル指標」を変化させる.

後付け条件は「テスト スイッチ」に影響を及ぼさないので，その使用はIFの使用と同じ価値をもってはいない.

8.1.3.3 多引数の並記

引数を受付ける命令で，8.2で紹介する各命令の構文規則は次の構造をとる.

命令語　1つのスペース　1つの引数

ここで引数が並記され得ることを明確に述べないときは，そこで示される引数の形式以外にこの命令に対して許容される引数構造はないことを示している．もし引数が並記され得ると記されておれば，次の規則を適用する.

Yを命令語，A1, A2, . . . , Anをそれぞれ1つの引数として，これらが構文規則で示された形をもっているとする.

いま構文規則が次のようであれば

Y␣A1

次の形の命令構文も許される.

Y␣A1, A2

Y␣A1, A2,　. . .　, An

(3つの点 . . . は，文字通りでなく，数が不定であることを示している．ただし有限個である．引数の間にはスペースはなく，コンマがあるだけである.)

Y␣A1, A2,　. . .　, An

は次のものと全く同じである.

Y␣A1␣Y␣A2␣　. . .　␣Y␣An

すなわち命令語 Y の後の引数を分離しているコンマは，次のものと定義的に同一解釈される．

␣Y␣

〔Port: 行上の文字数の制限は行がルーチンの交代時に，実際どのような字数をとるかによって起こってくる問題であり，“,” に対して各々を ␣Y␣ で置き換えたあとの行の字数で制限されるものではない．: Port〕

8.1.3.4　引数の後付け条件

命令 Do, Goto および Xecute のみに，引数の後付け条件が許される．従って以下の論議は，これら 3 つの命令にのみ適用される．

命令語または並記引数のどれかに後付け条件があるかどうかに関係なく，どの引数にも後付け条件を応用することができる．後付け条件をとる引数の解釈は以下のようになる．

1. 次に述べる理由によって，命令語は後付け条件をとっているのではないと仮定しうる．すなわちもし命令語が後付け条件をもち，その評価が偽であれば，もはや引数の解釈は必要なく実行されない．もし命令語が後付け条件を持ちその評価が真であれば，各々の引数はすべてあたかも命令語が後付け条件をもたないかのごとく，順々に解釈され実行に移される．

2. いま Pi を引数 Ai の後付け条件とする．ある引数の明確な実行が起こるか否かの疑問に関して言えば

   Y␣A1 : P1

   は

   Y : P1␣A1

   として解釈される．

   拡張型の引数並記を持つ

   Y␣A1, A2,　. . .　, An

   で，もし引数のどれかが，次の形であるなら

Ai : Pi

コンマをとった場合の拡張型は次のようになる.

Y : Pi␣Ai

3. 副次的効果について言えば，次の面で事態は複雑なものとなる．すなわち後付け条件とそれがついている引数が評価されてしまうまで，正常な実行順序（第6章参照）に対して効果を持たないということがあるからである．したがって

DO␣X : A

の実行においては，たとえ A が偽で，そのため サブルーチン呼出しが実行されない としても，Do が偽の値をとる 後付け条件を 持たない限り，入口参照 X と式 A は常に完全に評価されている．したがって

XECUTE␣^(3, 4) : Ø

は「略式グローバル指標」を変えようとする効果と，指定された値を取り出そうとする効果をもちながら（もしそれぞれの引数が定義されていないなら，このような試みは エラーになるのであるが)，しかし評価した式は実行しない.

8.1.3.5　引数間接指定

次の記述は引数間接指定が許されている命令に適用される．引数間接指定を許すどんな命令にも，引数の並記が許されている.

間接指定呼び出し

@<u>式子</u>

これは並記引数中の引数，また孤立した引数と置き換えることができる.

したがって次の構文

Y␣A1

が許される場合

Y␣@<u>式子1</u>

が許され，また

Y␣A1, A2, ... , An

という構文が許されるなら

Y␣A1, A2, ... , @式子 i, ... , An

も許される.

式子 i の値は，Yのとる構文規則を満足する引数または引数並記となる．このように Y␣@式子 1 における式子 1 の値は 1 個の引数

A1

であり得るし，引数並記

A1, A2, ... , An

でもあり得るし，または 1 つまたは複数の間接指定呼出しを含む

A1, A2, ... , @式子 i, ... , An

という引数並記でもあり得る．これはさらに続いて間接指定評価を行う必要がある．以上 3 種の可能性は，引数並記中のどこにでてくる@式子 i に対しても等しく有効なものである．理由は

Y␣A1, A2, ... , @式子 i, ... , An

の拡張型が次のようになるからである.

Y␣A1␣Y␣A2␣...␣Y␣@式子 i␣...␣Y␣An

8.1.4 命令の定義に共通する要素

8.1.4.1 READ と WRITE の中の書式

書式が READ または WRITE の部分として実行されるとき，書式制御出力動作をとる．複数の書式制御パラメータは左から右の順に一度に 1 つだけ実行される.

```
書式 ::= | | ! | ⎡!⎤        |
         | | # | ⎣#⎦ ... 〔?整数式〕 |
         |     ?整数式           |
```

書式制御パラメータは次の形をとる.

！は“左端復帰改行”動作を現用装置に起こさせる．

　(現用装置の記述は5.4を参照.）！の効果は ASCII 装置に CR LF を出力することに等しい．

　また CX は0に設定され，CY に1が加えられる（5.5参照)．

＃は“改頁”動作を現用装置に起こさせる．＃は ASCII 装置に CR FF を出力することに等しい．

　また CX, CY ともに0に設定される．現用装置がテレビ端末であれば，画面は空白となり，カーソルは左上すみに置かれる．

？整数式は“第整数式値番目の欄にタブレータをおく”と同じ効果を出す．より正確に言うと，もしCXが整数式の値より大きいか等しい場合は効果はなく，CX が整数式の値より小さい場合はスペースを（整数式 － $X）個出力するのと同じである（$X は CX の値である)．行の最も左の欄には数字0が与えられていることに注意．

もし現用装置が出力を受けない場合，書式の実行は CX と CY に書き込むこと以外に効果はない．

8.1.4.2　LOCK, OPEN, READ の時間制限

Lock, Open および Read 命令は，実行される命令の中に時間制限を書き込むことによって，時間制限仕様を任意選択事項として使うことができる．

時間制限　::=　:数式

これらの3つの命令には命令の実行が関連する特殊な外部条件がそれぞれ定義されている．

LOCK　　Lock は「下位排他規則」に違反した場合は実行されない（4.2参照)．すなわち Lock 命令の引数が示すそれぞれの節指定子と他のすべてのパーティションの「Lock リスト」全体との間に，何ら下位関係のないことが条件である．

OPEN　　Open は「装置排他規則」に違反した場合は実行されない（4.3参

照). すなわち命令の引数が示すそれぞれの装置指定子とシステムの他のすべてのパーティションの「Open リスト」全体との間に何ら共通な装置の存在しないことが条件である.

READ　READ の実行はメッセージの入力が終了したときに完了すると定義されている. すなわちメッセージの入力が終了したか否かということが条件である.

もし任意の時間制限が命令の引数に含まれないときは命令の実行は, 命令に関するこれらの条件が満足されたときにのみ進行する. もし条件が満足されないなら, 条件が満足されるまで実行は待たされる. そして満たされたとき実行は進行する.

もし任意の時間制限が命令の引数の中にあるときは, 数式は負でない値をもつ必要がある. もし数式が負ならばØとみなされる. いまtをこの負でない数値とすると, 時間制限はt秒間を越えない実行の停止を指定する. それは次のように説明できる.

もし, t=Ø ならすぐに前述の条件が試される. 条件が真なら「テスト スイッチ」は1になる. それが偽なら「テスト スイッチ」はØになる. 実行は遅れることなしに続く.

もし, tが正なら実行は条件が真になるまで停止されるが, どんな場合でもt秒より長く停止させることはない. もし実行が再開されるとき条件が真なら「テスト スイッチ」は1になり, そうでないときは「テスト スイッチ」はØになる.

もし任意の時間制限が存在した場合, 上に述べられた規則で条件が満足しないでも強制的に実行に移される場合, 命令の種類に従って次のように解決される.

LOCK　時間制限をもつ引数によって指定された機能は, 一部分しか実行されない. (すなわちパーティションの「Lock リスト」が空のままとなる.) そして制御は次の命令または引数に移る.

OPEN 時間制限をもつ引数によって指定された機能は実行されない．そして制御は次の命令または引数に移る．

READ 実行を再開する前に入力文字がどのようなものであっても，たとえそれが空のメッセージであっても，そこまでをすべての入力メッセージとしてみなす．実行の再開後に，かつ次の READ の実行の前に，入力される文字の処置に関しては言語装備者によって指定される．

8.2 命令の定義

8.2.1 BREAK

構文規則

BREAK | 〔␣〕 / ␣言語装備者によって指定される引数構文 |

詳 解

1. 命令語は B の1文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 言語装備者は引数をもつ形があるかどうかを指定し，引数をもつ場合には引数を並記できるかをも指定する.
4. 言語装備者は引数が後付け条件をとるかどうかを指定する.
5. 言語装備者は引数の間接指定が許されるかどうかを指定する.

実 行

1. 引数をもたない形. BREAK の実行の直接の結果としては，パーティション記憶域にも，システム記憶域にも変化はない. 実行は装置からの信号を受けるまで停止される. 信号の性質は信号を発する装置と共に言語装備者によって指定される.（実行が停止されている間に外部からの干渉の結果として，記憶域の内容は変化し得る.）
2. 引数をもつ形. 実行の仕方は言語装備者によって指定される.

例

1. B　　　後付け条件のない BREAK
2. B:X=3　　もし X=3 なら BREAK
3. SET␣X=^A(3)␣IF␣X=0␣BREAK␣␣KILL␣^A

   プログラマが作業状態を調べるために，グローバルを消す前に BREAK が入っている.

8.2.2 CLOSE

<u>構文規則</u>

CLOSE␣<u>装置指定式</u>〔：<u>装置パラメータ</u>〕

<u>装置指定式</u>　::=　<u>式</u>

<u>装置パラメータ</u>　::=　| <u>式</u> / (<u>式</u>〔：〔<u>式</u>〕〕　...　) |

<u>詳　解</u>

1. 命令語はCの1文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

<u>実　行</u>

このパーティションに対応する「Open リスト」の値は変化する．装置指定式の値が現用装置を意味するか否かによって「現用装置指定子」の内容は変化する.

言語装備者が規定する仕様によっては，装置指定式の値は，装置を示す装置指定子であったり，なかったりする．引数の意味する装置指定子が「Open リスト」中にある装置指定子と同じ装置を指定する場合は，これらの指定子は「Open リスト」から除外される．OPEN (8.2.12) の装置パラメータに関する記述を参照すること.

<u>前後参照</u>

4.3 「Open リスト」と<u>装置指定子</u>

5.4 「現用装置指定子」と「主装置慣例」

<u>*例*</u>

1. C␣3　　　　　　装置3を CLOSE する.
2. C:X=1␣X　　　X=1 のときに限り，装置 X を CLOSE する.

8.2.3 DO

構文規則

DO␣入口参照

詳 解

1. 命令語はDの1文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記することができる.
4. 引数は後付け条件をとることができる.
5. 引数の間接指定は許される.

実 行

Do の引数の実行は6.2の4. のシステム モデルの説明で詳述されている. 実行はサブルーチン呼出しの効果をもつ. サブルーチンへの入口の位置は引数が定義する. サブルーチンからの出口の位置は, FOR の範囲内にない QUIT または eor の実行によって決まる. しかしこれらの出口は, 続いて実行される Do または XECUTE に対する出口の位置としては作用しない.

前後参照

6.2 の4. Do 実行の順序

6.2 の2. FOR の範囲内にない QUIT および eor の実行

第7章 入口参照の構文規則と解釈法

例

1. DO␣X␣WRITE␣Y

   X とラベルされた行で始まるサブルーチンが実行され, その後変数 Y が書き出される.

2. DO␣X, Y

   DO␣X␣DO␣Y と同じ.

3. DO␣^PGM␣WRITE␣Y

   ルーチン PGM が呼び出される. その後 Y が書き出さ

れる.

4. DO␣A, ^PGM, ^ROU

多数のルーチン名やラベル名は引数として並記できる.

5. DO␣INT^JTO

ルーチン JTO を呼び出し, ラベル INT から始める.

6. D:X=1␣A^PGM, B, ^TEST

並記されたサブルーチンは X=1 のときだけ呼び出される.

7. DO␣^CEN:'A, B

ルーチン CEN は A=Ø のとき呼び出され, サブルーチン B は A の値に関係なく呼び出される.

8. D␣@X

D␣@X^PGM

D␣^@XYZ

D␣@A^@B

D:@A=B␣@C^@D:@E=@F

Do 命令の中で間接指定を使うことができる訳注1).

訳注 1) D␣LINE+3^ROUTINE の間接指定としては,
D␣@A+@B^@C および
D␣@X の形が考えられる. 但し
A="LINE", B=3, C="ROUTINE", X="LINE+3^ROUTINE".

8.2.4 ELSE

構文規則

ELSE〔␣〕

詳 解

1. 命令語はEの1文字に省略できる.
2. 命令語は後付け条件をとることができない.

実 行

ELSE の実行は6.2の4.のシステム モデルに関連して説明してある. ELSE が実行され そして「テスト スイッチ」($T) が値1をとるならばその行のうち eol を含まぬ残りの部分は実行されない. FOR の範囲内に ELSE がある場合その ELSE は実行されるごとに, 前に実行された結果とは無関係に, 独立的に解釈される.

前後参照

6.2 の3. 実行の解説

5.3 「テスト スイッチ」

例

次の2つの例は同じ効果をもつ.

```
IF␣X=1␣WRITE␣"YES"
ELSE␣␣WRITE␣"NO"

WRITE:X=1␣"YES"␣WRITE:'(X=1)␣"NO"
```

8.2.5 FOR

構文規則

FOR␣ローカル変数 = forパラメータ〔, forパラメータ〕...

forパラメータ ::= | 式 / 初値-増分パラメータ：数式3 / 初値-増分パラメータ |

初値-増分パラメータ ::= 数式1：数式2

詳　解

1. 命令語はＦの１文字に省略できる．
2. 命令語は後付け条件をとることができない．
3. 引数は並記することができない．しかしforパラメータは１引数内で並記できる．
4. 引数は後付け条件をとることができない．
5. 引数の間接指定は許されない．またforパラメータのレベルではいかなる間接指定も許されない．

実　行

FOR 命令の“範囲”は同行上のその FOR の次の命令から同行の eol のすぐ前までである．

FOR は指定されたローカル変数が順次与えられる値に対応して，それの範囲が繰返し実行されるように制御をする．各forパラメータは指定された順でローカル変数に値を与え，これらの各々の値に対してその範囲を一度ずつ実行させる．連続するforパラメータは順に左から右に上記の過程を制御する．ローカル変数の中にある式はどのようなものでも，例えば添字とか間接指定の式であっても，forパラメータを最初実行する前に，FOR 全体の実行１回に対して，一度だけその値が評価される．

次に各々の形のforパラメータがどのようにローカル変数の値と FOR の範

囲の実行の順序を指定し制御するかを説明する．以下で変数名 A, B, C は各 FOR 命令の制御に対してのみ用いられる隠された記憶レジスタであると考えてもらいたい．

1. もし for パラメータが式の形をとるならば
   a．Set ローカル変数 ＝ 式 とおく．
   b．範囲を一度実行する．
   c．この for パラメータの実行が完了する．
2. もし for パラメータが初値-増分パラメータの形をとるならば
   a．Set A ＝ 数式１ とおく．
   b．Set B ＝ 数式２ とおく．
   c．Set ローカル変数 ＝ A とおく．
   d．範囲を一度実行する．
   e．Set ローカル変数 ＝ ローカル変数＋B とおく．
   f．d に行く．

   この手順のままで行けば，無限循環になることがわかるであろう．この循環を断ち切るには FOR の範囲内にある QUIT または GOTO が実行されなければならない．この２つの循環終結法は，いかなる FOR の範囲内でも利用でき，その一般様式は下記のようになる．またこの形のものが１つあれば右側に別の for パラメータを置いても，もはやそれには制御機能はないことに注意しなくてはならない．

3. もし for パラメータが 初値-増分パラメータ：数式３ で，増分パラメータ（すなわち数式２）が負でない値とするならば
   a．Set A ＝ 数式１ とおく．
   b．Set B ＝ 数式２ とおく．
   c．Set C ＝ 数式３ とおく．
   d．Set ローカル変数 ＝ A とおく．
   e．もし ローカル変数 ＞ C ならば，この for パラメータの実行は完

結される.

f. 範囲を一度実行する.

g. もし ローカル変数 ＞ C－B ならば，この for パラメータの実行は完結される.

h. Set ローカル変数 ＝ ローカル変数＋B とおく.

i. f に行く.

4. もし for パラメータが 初値-増分パラメータ：数式3 で，増分パラメータ（すなわち数式2）が負の値をもつならば

a. Set A ＝ 数式1 とおく.

b. Set B ＝ 数式2 とおく.

c. Set C ＝ 数式3 とおく.

d. Set ローカル変数 ＝ A とおく.

e. もしローカル変数 ＜ C ならば，この for パラメータの実行は完結される.

f. 範囲を一度実行する.

g. もしローカル変数 ＜ C－Bならば，この for パラメータの実行は完結される.

h. Set ローカル変数 ＝ ローカル変数＋B とおく.

i. f に行く.

もし1つの行に2つ以上の FOR が存在するなら，これらの実行は入れ子になっている（ネスティング）と言われる．そして右の FOR は左の FOR に内包されているとか，“内側”であるという．2つの FOR がこのように入れ子(ネスト)になっているとき，外側の FOR の範囲を1回実行することは，内側の FOR の for パラメータ並記を全部通過して内側の FOR 命令全体が1回実行し終えることを意味する．(途中で GOTO または QUIT に出会って実行が終了することはあるが.)

1つの FOR の範囲にある何らかの命令が1回実行されたということは，そ

のときその命令がこの FOR のもつ for パラメータの1つに確実に制御されたことである．QUIT と GOTO の2個の命令は，それが1つの FOR の範囲内で実行されるときには，特別の効果をもつ．この効果を次に述べる．

FOR の範囲内での QUIT の実行は2つの効果をもつ．

a．それは QUIT に出会うとこの範囲の特定の実行を終わらせる．すなわち QUIT の右の命令は実行されない．

b．最も内側の FOR の範囲内で QUIT が起こる場合，この FOR の範囲を制御していた for パラメータの残りの値や，同じ for パラメータ並記内に残された for パラメータで指定されるはずの値はすべてこれ以上計算されず，それらの制御下にある FOR の範囲も実行されない．

言いかえれば，ある FOR の範囲内で QUIT が実行されると，これが1番内側の FOR の場合 FOR の実行を直ちに停止させ，もしこれに次の外側の FOR が存在するなら，この QUIT の実行はその外側の FOR の範囲の1つの実行を直ちに停止させる．

FOR の範囲内にある GOTO の実行は，同じ行の GOTO の左の<u>すべて</u>の FOR を直ちに停止させ，制御を指定された点へ移動させる．

<u>前後参照</u>

6.2 の7．FOR の実行

6.2 の2．FOR の範囲内の QUIT の実行

6.2 の5．FOR の範囲内の GOTO の実行

8.2.6 GOTO の定義

8.2.13 QUIT の定義

<u>例</u>

1. F␣I=1：1␣WRITE␣I␣Q：I＞2␣␣W␣"＊"

出力は 1＊2＊3

2. F␣I=1：1：3␣W␣I

出力は 123

3. F␣I=3：−2：−2␣W␣I

出力は 31−1

4. F␣I=5,3.4,7,9␣W␣I

出力は 53.479

5. F␣I=.4,1：2：5,9,10：1␣DO␣A␣IF␣I>15␣QUIT

A が 12 回実行される.

6. F␣I=1：1：2␣F␣J=2：2：6␣W␣I, "@", J, "*"

出力は 1@2*1@4*1@6*2@2*2@4*2@6*

7. F␣I=1：1：3␣F␣J=10：10：30␣Q：I*J>30␣␣W␣I, "@", J, "*"

出力は 1@10*1@20*1@30*2@10*3@10*

8. F␣I=3：5：0␣WRITE␣I

何も書かれない. I の最終値は 3 である.

9. FOR␣I=.01：.0001：.02␣D␣A

これは A を 101 回繰り返す.

10. F␣I=X：Y：Z␣D␣A

11. F␣I=1：2：10␣SET␣I=I−1␣W␣I

指標として用いられるローカル変数の値を変えることは許される. 出力は 0123456789

12. SET␣Z=10␣F␣I=2：2：Z␣SET␣Z=Z−1␣DO␣A

ループを終わらせる値は, 範囲の実行される前に計算してしまうので, A は 5 回実行される.

13. F␣I="TEST", X, 3：4：5 ・・・

式のみの形の for パラメータが使われるとき, その式の値として非数値データを用いてもよい.

14. F␣I=X：Y：Z␣F␣J=A：B：C␣F␣K=1：1：3 ・・・ G␣LCS

GOTO があると入れ子になったすべての FOR から出てしまう.

15. F␣@A=1 : 1 : 3␣DO␣B

間接指定は for パラメータ全体には使えないが，普通の用法で変数名の代わりに使うことはできる.

16. F␣I=@X : 1 : 3␣DO␣@B

17. F␣@I=@X : @Y : @Z␣DO␣@B

8.2.6 GOTO

構文規則

GOTO␣入口参照

詳　解

1. 命令語はＧの１文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができる.
5. 引数の間接指定は許される.

実　行

GOTO の引数の実行は6.2の5．のシステム モデル の説明で明らかである．これの実行は単純な制御の移動である．実行される引数が他のルーチン名であるときは，パーティション スタックの 最上部の ルーチン体が，この呼ばれたルーチン体にとって代わられる．DO と違って引数が並記されていても，実際に制御を移動させるものは，せいぜい１個の引数であることに注意.

前後参照

6.2 の5．実行の解説

第７章　構文規則と入口参照の解釈

8.2.5　FOR の範囲内の GOTO の実行

例

1. GOTO␣XYZ

   実行はこのルーチン内の XYZ と名付けられた行の始まりに続く.

2. GOTO␣^PGM

   実行の流れは GOTO を含むルーチンから，ルーチン PGM の最初の行に切りかわる.

3. G␣CC+2^NIH

実行の流れはルーチン NIH に切りかわり，CC と名付けられた行の後の 2 番目の行から始まる.

4. G␣13^GRAY : A=1

A=1 のとき実行の流れは，ルーチン GRAY に切りかわり 13 と名付けられた行から始まる.

5. G : A=1␣13^GRAY

例 4. と同じ.

6. G␣^ADM: A=Ø, X^CEN

A=Ø のとき実行の流れはルーチン ADM に切り換えられ，その最初の行から始まる．A=Ø でないなら実行の流れはルーチン CEN に切りかわり，X と名付けられた行から始まる.

7. G : A=1␣AAA : B=2, 13 : D=4

この行は次の行と同じである.

G␣AAA : A=1 & (B=2)␣G␣13 : D=4 & (A=1)

8. G␣@X

G␣^@B

G␣A^@B

G␣@A^B

G␣@A^@B

GOTO 命令内では間接指定を使う多くの機会がある.

9. G : X=@Y␣@A+@C+D^@PGM : @E=@F

もし制御が移動したとすれば，実行の移る行はラベル @Aのあとの @C＋D 番目の行となる.

8.2.7 HALT

構文規則

HALT〔␣〕

詳　解

1. 命令語はHの1文字に省略できる.
   HALT が省略されるとき, 引数のないことで HANG と区別されること
   に注意.
2. 命令語は後付け条件をとることができる.

実　行

この命令のあるパーティションの過程は実行を停止する.

このパーティションに関係するシステム記憶域は次の状態となる.

1. このパーティションに対応する「作業番号 P-ベクトル」の要素は 0.
2. このパーティションに対応する「Lock リスト P-ベクトル」の要素は空.
3. このパーティションに対応する「Open リスト P-ベクトル」の要素は空.

前後参照

6.2 の 1. HALT の実行

例

1. H　　　　　　無条件 HALT
2. H:A=1␣␣WRITE␣"TEST"
   A=1 なら過程は停止 (HALT) する. そうでない場合の実行は続けられ TEST と書かれる.
3. H:A=@B
   後付け条件で間接指定を用いることができる.

8.2.8 HANG

構文規則

HANG␣整数式

詳 解

1. 命令語は H の 1 文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記することができる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実 行

t を引数の値とする. もし t が 0 または負なら HANG は効果をもたない. もし t が正なら実行は t 秒間停止される. HANG の実行の直接の結果として記憶が変化することはない.

前後参照

例

1. A IF␣$P($H, ",", 2)<X␣HANG␣3␣G␣A

   真夜中からの秒数が X より少ない場合, 実行は 3 秒停止したのち行が繰り返される.

2. H:I=3␣10␣WRITE␣!

   I=3 ならば, WRITE 命令の実行されるまえに, 10 秒間の HANG が実行される.

3. H␣@X

   H:$E(@A, B, @C)␣X+@Y+Z

   HANG 命令で間接指定を使うことができる.

8.2.9 IF

構文規則

引数をもたない形式：　IF〔␣〕

引数をもつ形式：　IF␣真偽値式

詳　解

1. 命令語はＩの１文字に省略できる.
2. 命令語は後付け条件をとることができない.
3. 引数は並記できる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実　行

IF の２形式とも次のように統一して考えることができる．すなわち実行は２つのステップからなり，引数をとらない形式では，第１のステップは実行されず，第２のステップは２つの形式に共通であると考える.

ステップ 1.　引数の値の真偽値判断は「テスト スイッチ」におかれる.

ステップ 2.　「テスト スイッチ」の内容が１なら反応はなく，次の命令または引数の実行が続く．もし「テスト スイッチ」の内容がØなら，この引数の右にある行上のすべての文字の実行は eol の手前まで禁じられる．このような禁じられた実行は直接的にも間接的にも何に対しても影響をもたない.

前後参照

6.2 の３．IF と ELSE の実行

8.2.4　ELSE の実行

8.1.3.2　後付け条件と IF との相違

例

1. I␣X=1␣WRITE␣X

IF 命令の右のコードは X=1 のときだけ実行される.

2. I␣Y␣S␣Z=3

IF 命令の右のコードは, Y の数値評価が0でないときだけ実行される.

3. I␣X=1, Y=2␣DO␣3

DO␣3 を実行するかしないかについては次の2つの形とも同じ条件下では同じ結果になる.

IF␣X=1␣IF␣Y=2␣DO␣3

IF␣X=1 & (Y=2)␣DO␣3

すなわち並んだ引数の連続実行の効果は, <u>AND</u> が含まれているのと同じ結果になる.

4. I␣X, ^A(Y)␣G␣3

しかし副次的効果が異なることもあり, これは

I␣X & (^A(Y))␣G␣3

と同効果ではない. それは後者が必ず ^A(Y) を評価するが, 前者では必ずしもそうではないからである.

5. I␣X?1N2A!(Y=2)

IF 命令に対してしばしば複雑な式が引数となる.

6. IF␣X=1␣W␣!, "TEST"

ELSE␣␣W␣!, "EXAM"

IF␣␣W␣"␣ARGUMENTLESS"

もし X=1 なら TEST␣ARGUMENTLESS と書き, そうでないときは EXAM と書く.

7. IF␣@X

IF␣A, @B, C

IF␣A=@B, @C=D, E+@F+H

IF 命令で間接指定を使うことができる.

8.2.10 KILL

構文規則

1. “全面的 KILL” すなわち引数をもたない形式：

KILL〔␣〕

2. “選択的 KILL” の形式： KILL␣記憶域参照
3. “排他的 KILL” の形式： KILL␣(名前〔, 名前〕...)

詳 解

1. 命令語は K の 1 文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる. 引数並記では引数をもつ形式の 2 種を混合することができる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実 行

KILL の 3 形の実行は単元操作の形を用いて説明できよう. すなわち 1 つの変数のみを KILL することを考える. V を 1 つの記憶域参照とする. V に対応する「名前」属性をもった節が「名前付きパーティション記憶域」ないし「名前付きシステム記憶域」内にあった場合それを N とする. (この対応の説明は 12.4 にある.)

V を KILL することは次の意味をもつ. もし N が存在しないなら効果はない. N が存在するなら, N とすべての下位節は消される. N の上位節の属性は V が消されても変化しない.

1. “全面的 KILL” の実行はすべてのローカル変数を消す.
2. “選択的 KILL” の実行は引数で指定された変数を消す.
3. “排他的 KILL” の形式の説明中各々の名前は添字を含まないローカル変数 1 個を意味する. 引数で指名された 1 つないし複数のものを除いて, 添字をもたないローカル変数はすべて消される. また結果としてローカ

ル変数の下位節は，引数で指名された変数の下位節を除いてすべて消される．

<u>前後参照</u>

4.1 「名前付きシステム記憶域」の構造

5.1 「名前付きパーティション記憶域」の構造

12.4 <u>記憶域参照</u>

<u>例</u>

1. KILL␣X

ローカル変数 X が消される．

2. KILL␣X, Y, ^A(3, 4)

ローカル変数 X, Y とグローバル変数の ^A(3, 4) およびこれらの下位節が消される．

3. KILL␣(A)

A およびその下位節を除いてすべてのローカル変数が消される．

4. K␣(A, C, DEF, %1)

これらの名前をもつ変数とその下位節を除いてすべてのローカル変数が消される．

5. K␣␣SET␣X=$T

引数なしの KILL はすべてのローカル変数を消す．この行の実行後，X は定義された値をもつただ1つのローカル変数となる．（ただし $T が定義されている場合.）

6. K:X=1␣^A(X)

X=1 のとき ^A(1) とその下位節が消される．KILL はどの形でも命令語に後付け条件をとりうる．

7. K␣^A

グローバル配列 ^A が完全に消される．

8. K␣@X

8.2.10 KILL

K␣A, @B, C

K:@B=C␣A(B, @C)

KILL 命令では間接指定を使うことができる.

8.2.11 LOCK

構文規則

1. 引数をもたない形式：

   LOCK〔␣〕

2. 引数をもつ形式：

   LOCK␣ | 変数名 / (変数名〔, 変数名〕 ･･･) | 〔時間制限〕

詳　解

1. 命令語は L の1文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができる.
5. 引数の間接指定は許される.

実　行

上の引数形式 LOCK␣変数名〔時間制限〕は，下の引数形式 LOCK␣(変数名)〔時間制限〕で変数名が1つの場合と等価となる．それゆえに以下の定義では一般的な引数形として，第2の形式のみを取り扱うことにする.

LOCK の2つの形式の実行は以下のように統一して考えられる．実行に2つのステップがある．第1のステップは2つの形に共通である．第2のステップは引数なしの場合には実行されない.

ステップ 1.　パーティションに対応する「Lock リスト P–ベクトル」は無条件で空になる.

ステップ 2.　以下に述べる条件に従って,パーティションに対応する「Lock リスト P–ベクトル」の要素は新しい値をとる．以下の説明でこの過程を明らかにしよう.

8.1.4.2で，1つの命令引数に時間制限がついているときは，その特定の命令に関係する外部条件に合わせて実行がなされる，と述べた．LOCK に関する

条件が真となるのは，このパーティションに関係する現在の「Lock リスト」と引数の示す新しい「Lock リスト」とが置換されるときに，「下位排他規則」(以下「規則」，4.2参照) に違反しないときに限られる．言い換えるならば，この“規則”によって各パーティションの「Lock リスト」が互いに干渉しあうことが防がれている．LOCK 引数は (以下に述べる方法で) このパーティションのための新しい「Lock リスト」を定義する．その新しい「Lock リスト」は全般にわたって「規則」に違反してはならない．全く違反しないならば条件は真となる．もし「規則」に違反するのならば条件は偽となる．述べるまでもなく条件の真偽は時間の関数である．すなわち他のパーティションの「Lock リスト」も変化しているであろうから．

時間制限がない場合，このパーティションの「Lock リスト」に新しい値を，「規則」に違反せずに指定することができなければ，それができるようになるまで実行は中断して待っている．「Lock リスト」に完全に新しい値を入れることができたときは，それらをリストに入れて実行が進行する．「テスト スイッチ」は変更されない．

もし時間制限がある場合，実行の定義は上述の定義にさらに２つの付加要素を加えたものと同じである．t を時間制限要素である負でない値とする (8.1.4.2)．

1. t=0 のとき実行の一時停止は起こらない．t が正であれば実行は，必要な場合に限り一時停止する．しかし少なくとも t 秒たつと実行が強制的に再開される．(以下で，もし t=0 ならば“実行の再開のとき”は実行の始まるときと定義される.)
2. 実行の再開時に条件が調べられる．もし真なら「Lock リスト」はその指定された値をとり，「テスト スイッチ」は値１をとり実行が再開する．条件が偽ならば「Lock リスト」は空のままで「テスト スイッチ」は0となり実行が再開される．

引数が指定するリストの一部のみが「Lock リスト」内に部分的に割り当てられることはない．従って「Lock リスト」は一部分のみ先に，または遅れて変化することはない．もし LOCK を実行して必要な「Lock リスト」を確立できないなら，もう一度明確に LOCK 命令を実行し直さなければならない．

引数によって指定される「Lock リスト」は引数が含む変数名と同じだけの要素を含む．これらは1：1の対応をする．引数の各々の変数名に対応する「Lock リスト」の要素はその変数名から得られる<u>節指定子</u>である．<u>変数名</u>の構文規則と節指定子を得るアルゴリズムについては 12.4 を参照のこと．

LOCK の引数は名前であって「名前付きパーティション記憶域」または「名前付きシステム記憶域」についての参照ではないので，次のような性格をもつ．

1. LOCK の引数としての変数名は「名前付きパーティション記憶域」ないし「名前付きシステム記憶域」内にそのような節が存在するかどうか，またその節がどのような属性をもつかということには無関係である．
2. LOCK の引数としてグローバル変数があっても「略式グローバル指標」には影響を及ぼさない．

<u>前後参照</u>

4.2 「システム記憶域」の中の「Lock リスト」
5.3 「テスト スイッチ」
8.1.4.2 時間制限
12.4 「節指定マッピング」，<u>変数名</u>

<u>例</u>

1. LOCK ␣ ^A

この LOCK のあとで，実行が再開される時点からと，このパーティションで次の LOCK が実行される時点までの間は，他のプログラムで添字付きでないグローバル名 ^A またはその<u>名前</u>部分が A である添字付きグローバル名を含む LOCK の引数を実行することは不可能となる．なお

このプログラムで前に LOCK された変数名は LOCK を解かれる.

2. L␣(^B(1, 2), ^C(3))

この LOCK のあとで実行が再開される時点から，このパーティションで次の LOCK が実行される時点までの間は，他のプログラムから次の変数のどれも LOCK することができない.

^B　^B(1)　^B(1, 2, 3)

^C　^C(3)　^C(3, 4, 5)

しかしながら以下の変数は他のプログラムから LOCK することができる.

^B(2)　^B(1, 3, 4)

^C(4)

3. L␣␣SET␣X=1

引数なしの LOCK 命令はこのパーティションの前の LOCK を解除する.

4. L␣D(1):3␣G␣A:'$T

LOCK␣D(1) について 3 秒間試みられる. 試みがその間に成功したら，$T の値は 1 をとる. そうでないなら $T は 0 となり，実行は A のラベルをもつ行から続行する.

5. L␣D(1) : 3␣ELSE␣␣G␣A

上の 4 と同じ.

6. L:X=1␣^EF(^A(5))

LOCK 命令は後付け条件をとることができる. グローバル変数名があってもこの例のように，評価される式として指定されるとき以外は，「略式グローバル指標」に影響を及ぼさない. ここではそれは添字である[訳注1]. 他のこのよう

な可能性は後付け条件と間接指定で起こりうる.

7. L␣@A

   L:@B=C␣X(@A), @B

   LOCK 命令では間接指定を使うことができる.

訳注 1) この LOCK 命令の実行の結果,「略式グローバル指標」の値は ^EF ではなく ^A となる. ただし, ^A(5) が定義されていない場合はエラーであり,「略式グローバル指標」の値は "未定義" となる.

8.2.12 OPEN

構文規則

OPEN␣装置指定式 〔 : | 装置パラメータ〔時間制限〕 / 時間制限 | 〕

装置指定式 ::= 式

装置パラメータ ::= | 式 / (式〔:〔式〕〕 ...) |

詳　解

1. 命令語はＯの１文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実　行

命令引数中の装置指定式は入出力“装置”を示す．入出力装置とは連続的な文字の出口および/または入口として作動するものである．この式の値は4.2で述べたようにこのパーティションに対応する「Open リスト」中の装置指定子として使われることになる．この式の値が満足すべき構文上の規則を指定するのは言語装備者である．また言語装備者はこの式のとりうる値と使用しうる装置一式との間の対応関係を指定する．

装置指定子Ｄがパーティションの「Open リスト」中にある場合に限ってこのパーティションが装置Ｄを占有すると言われる．

１つの引数をもつ OPEN の目的は１つの装置を占有することである．複数引数形式をとる場合は各引数の実行によって「装置排他規則」(以下「規則」)に違反しない限り，各々の引数の指定する装置指定子が順次パーティションの「Open リスト」中に加えられる．「規則」によると２つのパーティションは同

じ装置を占有することはできない．指定された装置指定子をパーティションの「Open リスト」に加えることが「規則」に反しない場合に限って OPEN 命令(8.1.4.2参照）に関する条件は真となる．

引数中に時間制限がない場合は，「規 則」に違反することを さける必要のある場合に限って実行は中断される．引数によって指定される装置指定子を「規則」に違反せずこのパーティションに関する「Open リスト」に加えうるときは,それが加えられて実行が進行する．「テスト スイッチ」は影響をうけない．

引数中に時間制限がある場合は，実行の定義は上述と同様で，その上に2つの付加的要素がある．ここに時間制限を意味する負でない値 t を考える（8.1.4.2参照).

1. t=0 のとき実行の中断は起こらない． t が正であるなら前記の規定によって必要であるときに限り実行が中断される．しかし t 秒目の終わりに実行は強制再開される．（以下で，もし t=0 ならば“実行の再開のとき”の定義については，実行が始まる時点として扱う.）
2. 実行が再開されるときは OPEN 命令に関する条件が検査され，もし真ならば装置指定子は「Open リスト」に加えられ,「テスト スイッチ」の値は1となり，装置パラメータは以下に指定されるように処理されて実行が進行する．もし条件が偽なら「Open リスト」は変化せず,「テスト スイッチ」には値0が与えられ，装置パラメータには影響を与えず実行が進行する．

該当する装置のために言語装備者が選んだパラメータはすべて引数の装置パラメータ部分を使って指定される．装置パラメータの各々の式の値として許される構文規則と各々の値の解釈については言語装備者が指定する．以上はすべての言語装備に共通であるが，装置パラメータはパーティションとは関係なく，装置に対してそれぞれただ一つ記憶されている．システムの始動時に各装置は言語装備の指定によって，適当な1組の欠損代用（default）パラメータを与えられる．OPEN，CLOSE または USE を実行するごとにその引数によって,

1つ以上の装置パラメータを変化させることができる．各装置パラメータの値は，どのパーティションが指定するかにかかわらず，次にくる OPEN, CLOSE, USE の引数の実行によってパラメータが置き換わるまで存続している．

前後参照

4.3 装置指定子

例

1. OPEN␣4

   装置4を占有できる．もし他のパーティションがすでに装置4を所有していたら，この過程は装置が利用できるまで一時中断される．

2. O␣4, X, Y

   上と同様に装置 4, X, Y が順に OPEN される．

3. O␣X::3␣G␣A:'$T

   3秒間装置 X を OPEN することを試みる．それが成功したとき $T は値1をとる．そうでなければ $T は0となり，実行は A と名付けられた行から続行する．

4. O␣X::3␣ELSE␣␣G␣A

   上の3と同じ．

5. O:X=1␣A, B, C

   OPEN 命令語は後付け条件をとりうる．

6. O␣A:("IO":300:"TTY"), B:("O":800:"MT":20):TIM

   ある種の言語装備では装置特有のパラメータを設定するために追加的な引数を用いる．

7. O␣@X

   OPEN 命令の引数に間接指定を使うことができる．

8.2.13 QUIT

構文規則

QUIT〔␣〕

詳 解

1. 命令語は Q の1文字に省略できる.
2. 命令語は後付け条件をとることができる.

実 行

QUIT は次の明確な2つの文脈中のどちらかで起こる.

1. FOR の範囲内
2. FOR の範囲外

FOR の範囲外の QUIT の実行作用は, DO または XECUTE の引数によって始まるサブルーチンからの出口を作ることである. (処理過程を起動したのはルーチン外の1つの DO の実行であったと考えてよい.) システム モデルに関して言えば, FOR の範囲外の QUIT の実行は, パーティション スタックから最上部をはじき出すことになる (6.2の2. 参照). もしこれがスタックを空にするなら HALT が実行されることになる.

eor の実行すなわちルーチン体の終結の実行は FOR の範囲外の QUIT の実行と等しい効果をもつ.

FOR の範囲内の QUIT の実行は, その範囲内の最も内側の FOR の実行を直ちに終わらせる. そして QUIT のある行の FOR がただ1つならば QUIT の実行は行の実行を終わらせる. もし行の FOR が2つ以上であれば QUIT の実行は最も内側の FOR を終わらせ, 内から2番目の FOR の範囲のその回の実行を終わらせることになる.

前後参照

6.2 の 2. QUIT の実行

8.2.3 DO

8.2.5 FOR

8.2.7 HALT

8.2.19 XECUTE

<u>例</u>

1. I␣X>3␣Q␣␣WRITE␣Y

2. Q:X>3␣␣WRITE␣Y

   IF の範囲が行の終わりまで及ぶからと言って，これら2例の効果は同じではない．最初の例では，WRITE␣Y は絶対に実行されない．

3. 以下のルーチン体を考えてみよう．

```
A  F␣I=1:1:100␣D␣B␣Q:I+X>6␣␣W␣I
   W␣"␣END"␣Q
B  S␣X=I*2␣D␣C␣W␣"B"
   Q
C  W␣"C"␣Q
```

このルーチン体では，

a．行 A の QUIT は FOR ループを終わらせる．

b．行 A+1 の QUIT はルーチン体を終わらせる．

c．行 B+1 の QUIT は実行を FOR ループの内側にもどらせるので DO␣B の実行を終わらせる．

d．行 C の QUIT は実行を行 B にもどらせるため DO␣C の実行を終わらせる．

出力は CB1CB2CB␣END である．

8.2.14 READ

構文規則

```
        | 定文字列                   |
        | 書  式                     |
READ␣   | ローカル変数 〔時間制限〕  |
        | *ローカル変数 〔時間制限〕 |
```

詳 解

1. 命令語は R の 1 文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実 行

READ の実行は現用装置の入力そして/または出力のもととなる. READ は引数の構文形にしたがってそれぞれ異なった規則によって実行される.

READ␣定文字列は 5.5 で解説したように CX と CY の値を変えながら現用装置に対して定文字列を出力する原因となる. 現用装置が出力を受けつけない場合の CX と CY に対する効果は 5.5 のとおりであるが出力動作はなされない.

READ␣書式は 8.1.4.1 で解説したように CX と CY の値を変えながら現用装置に対して書式制御情報を出力する原因となる. もし現用装置が出力を受けつけない場合の CX と CY に対する効果は 8.1.4.1 のとおりであるが出力動作はなされない.

READ␣ローカル変数〔時間制限〕は現用装置から入力された ASCII データ文字列を終結基準に出会うまで蓄積させる. この終結基準は以下に述

べるように時間制限があるかどうかに依存する.

言語装備者は，装置によって差異はあろうが“明確な終結手順”が特定の文字の入力で行われる場合には，手順の実行時にこの文字が入力文字列の一部となるかどうかも定義しておく.

tを時間制限に関する負でない値とする（8.1.4.2参照).

1. 時間制限がないときは，実行は直ちに中断され，明確な終結手順が行われるまで待っている．終結手順が行われると実行はただちに再開される．入力文字列とは，実行停止中に入ってきたすべての文字を受け入れ順に左から並べて連結したものである.
2. 時間制限が存在して t=0 なら，実行は停止されず，入力文字列は空となる.
3. 時間制限が存在しtが正なら，実行はただちに停止し，t秒後または明確な終結のいずれかの早い方によって再開する．入力文字列の値は，実行停止中に入ってきたすべての文字を受け入れ順に左から並べて連結したものである.

Iを上で定義した入力文字列の値とする．実行再開時に以下のことが起こる.

1. SET␣<u>ローカル変数</u>＝I　の実行.
2. 時間制限が存在しないなら「テスト スイッチ」は変化しない．時間制限が存在して t=0 なら「テスト スイッチ」は0となる．時間制限が存在してtが正で，明確な終結手順が実行の再開時またはそれ以前に入力されていないなら,「テスト スイッチ」は0となる．時間制限が存在してtが正で，明確な終結手順が実行再開時またはそれ以前に入力されていれば,「テスト スイッチ」は1となる.

READ␣*<u>ローカル変数</u>〔<u>時間制限</u>〕は1文字を読み込む．読み込まれる文字は ASCII 文字であってもよいし，そうでなくてもよい．例えば入力される

この“文字”を状態値と考えてもよい.

引数中に指名されたローカル変数は常に整数値を受け取る．整数値はコード表（例えば入力文字の2進法 ASCII コードを10進法表示にしたもの）の文字に関係させておくかどうか，またはその他にいくつかの判断を持たせるかどうかについては，言語装備者が装置独特の方法で定義しておくことができる.

1. 時間制限が存在しないとき実行は現用装置から1個の入力文字が入るまで中断する．入力されるとただちに実行は再開される.
2. 時間制限が存在して t=0 ならば，実行は中断されず，現用装置からどんな文字が入力されても考慮されない.
3. 時間制限が存在して t が正ならば実行はただちに中断され，実行は t 秒間経過するか，現用装置から1個の文字を受け取るか，のどちらか早い方で再開される.

実行再開時に次のことが起こる.

1. 時間制限が存在して t=0 か t が正であり実行の中断の間，文字が入力されないなら SET␣ローカル変数=－1 が実行される.
2. 文字の入力があれば SET␣ローカル変数=入力文字に関した整数値を実行する.
3. 時間制限が存在しないときは「テスト スイッチ」は変化しない．時間制限が存在して t=0 ならば「テスト スイッチ」は0となる．時間制限が存在して t が正で実行中断期間中に入力文字が入ってこないなら，「テスト スイッチ」は0となる．もし時間制限が存在して t が正で実行中断期間中に入力文字が入ってくれば「テスト スイッチ」は1となる.

READ␣ローカル変数〔時間制限〕の形（すなわち“*印のついてない”形）は，5.5で解説したように CX と CY に影響を与える．*印をもつ形も言語装備者の文字の解釈法によっては5.5に従って，CX と CY に影響を与え

る.

2つの形，READ␣ローカル変数〔時間制限〕と READ␣*ローカル変数〔時間制限〕の実行定義を厳密に読めば結局，これらの READ の実行前に，かつもし以前に他の READ のある場合はそれの実行後に到着した文字や明確な終結手順の入力は無視されることになる．このような解釈はある種の操作環境では望ましくない．そのために言語装備者は以下に示すような一貫した，しかし多分に装置依存的な解釈変更の方法のいくつか，またはすべてを採用してもよい．

1. 入力文字は列を作って入ってくるが，READ 命令はどの文字をも無視せずに到着順に取り込みその順に取り出して行く方法で，現用装置から受けとるものを扱ってゆく．*印のない READ では，現用装置からこの READ の実行前に，かつ以前のすべての READ の実行後に入ってきたすべての文字を左から右に並べて連結する．*印付きの READ では，それ以前のすべての READ の実行後にはじめて到着する1個の文字が処理される文字となる．
2. *印なしの READ の場合言語装備者は，明確な終結が多数入ってくる場合をどう扱うかについて決めておかなければならない．このために選択できる2つの方法がある．
   a. 入力文字は終結基準によって区分されて並んでいて，READ はこのように分割されている1つの区分のみを受け付ける．
   b. もし READ の開始のとき，一番最近の終結基準以後に1文字でも入ってきたら，READ 開始以前のすべての終結基準は無視される．もし READ 開始直前の終結基準後文字が何も入らなければ，その終結基準はこの READ 開始直後起こっていると考える．
3. 時間制限が存在するとき t=0 がその特別な場合であるのをやめ，t が正の値である場合と同様に，もしその READ 前かつそれ以前のすべての READ 後に試された条件（明確な終結基準ないし1文字の到着）が

確かに起こっているとき，値1を「テスト スイッチ」に入れる．ただし，「テスト スイッチ」に値1を割り当てるということは，ローカル変数に特定の条件によって値が与えられているということのみを意味しているのである．すなわち＊印のない READ の場合は明確な終結手順が実行されたということであり，＊印のある READ の場合はローカル変数には入力文字に対応した値が割り当てられており，欠損代用値（default value）－1を割り当ててはいないということである．

前後参照

8.1.4.1 書式

8.1.4.2 時間制限

12.2 定文字列

12.4 ローカル変数

例

1. READ␣X

   データ文字列がローカル変数 X に入る．

2. R␣X, Y, Z

3. R␣"PATIENT？", X

   メッセージ PATIENT？ が現用装置上に現われる．その後実行はデータが入るのを待ち，それが X に入れられる．

4. R␣!!, "FIRST？", X, ？30, "SECOND？", Y

   このコードは現用装置上で左端復帰と改行を2回起こさせ，メッセージ FIRST？ を表示する．データが変数に入った後，装置に左端から30文字分あけて，メッセージ SECOND？ が現われる．その後データが入れられ変数 Y に読み込まれる．

5. R␣"TIMING？", X：10␣G␣A：'$T

   R␣"TIMING？", X：10␣E␣␣G␣A

   これら2つの行は同じ動作をする．TIMING？ が現われた後，作業は入力を待つために10秒間一時中断する．もし左端復帰

キー（ここでは終結手順と仮定）がこの時間内に押されることがあれば，$T に1が割り当てられる．そうでないなら $T は 0 となり，X は部分的入力を入れ，実行は A から続けられる．もし何の文字も入ってこないなら，X の内容は空文字列となる．

6. R␣*X, *Y, *Z

言語装備者が用いることのできる1つの解釈法として，この形の READ は，単一文字の入力をその ASCII 数字コードへの変換に使いうる．もし文字の A, B および CR（左端復帰）が続けて入力されれば結果として次の値が割り当てられる．

X＝65

Y＝66

Z＝13

7. R␣*X:10␣E␣␣G␣A

＊印形は時間制限を使うことができる．もし時間制限が1つの文字の入力もないまま終結するなら，X は値 −1 をもち，実行は A に続けられる．

8. R:$D(^A)␣!!, "TEST", X:12, *Y, *Z:10, #, "SAMPLE"!, A:9

READ 命令は後付け条件をとりうる．種々の引数形のすべてが，1つの引数並記の中に現われる．2個以上の時間制限が存在するときは，$T の値は最後に実行される（最も右の）時間制限の結果を入れている．

9. R␣@X

R␣X:@Y

R:@C␣@X:@Y

R␣*@A

READ 命令は間接指定と共に用いてよい．

8.2.15 SET

構文規則

SET ␣ | 記憶域参照 | = 式
| (記憶域参照 〔, 記憶域参照〕 . . .) |

詳 解

1. 命令語は S の 1 文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる. (上に示したのは引数が 1 個の形である.)
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実 行

SET 命令はデータ文字列を「名前付きパーティション記憶域」または「名前付きシステム記憶域」の 1 つ(またはそれ以上)の節の「値」属性の中におく手段である. データ文字列は式の値である. すなわち節 (複数節) は記憶域参照 (複数記憶域参照) によって指定される. (括弧の中の語は括弧に入った引数形に適用される.)

実行は次の順序で起こる.

1. 記憶域参照 (複数) の中で間接指定または添字は引数に現われた順番で左から右に向かって評価される.
2. 式が評価される. V を式の値と仮定しよう.
3. 各々の記憶域参照 (もし数個あるなら左から右の順に一度に 1 つ) によって規定される「名前付きパーティション記憶域」または「名前付きシステム記憶域」の節の「値」属性として V が割り当てられる. (記憶域参照から節指定子に対してなされるこのマッピングの詳細については 12.4 で述べられる.) ここでは次のことを考察しておくのがよい.

a. 記憶域参照が略式グローバル参照であれば, 「略式グローバル指標」

の内容は，節指定子を作り出すために利用される．その結果できた節指定子は「略式グローバル指標」の内容に（恐らく新しい）値を与える．この過程は5.2と12.4で解説されている．

b．もし記憶域参照によって規定された「名前」属性をもつ節が存在しないならば，最少数の節が作り出され，その結果

(1) 記憶域参照によって定義された「名前」属性をもつ節が存在する．そして

(2) この節は，もし「名前」属性の最初の文字が“^”ならば「名前付きシステム記憶域」の，“^”がなければ「名前付きパーティション記憶域」のいずれかの登録節の下位節である．

c．いくつかの節が以上のようにして作り出されるとき，これらの「名前」属性は上に述べた生成過程に従って定義される．節の他の属性は以下のようにして定義される．

(1) 「名前」属性が記憶域参照によって定義される節には「D」属性として1，「値」属性としてVが与えられる．

(2) このようにして生成される他のすべての節には（そしてこれらはその直接上位節から始まる上位節の一連の鎖を形づくるのであるが）「D」属性として10が与えられる．もっともこれらの節の「値」属性がまだ無い間は何の値も入れておく必要はない[訳注1)]．

d．そしてまた，このような方法で節が造り出される場合，この生成の前には下位節を何も持たず，生成の後には生成されたすべての節を下位節としてもつような1個の独特な節が存在する．この節の「D」属性には以下の値が与えられる．

もしこの値なら	→	この値にかわる
0		10
1		11

訳注 1)　すなわち，「値」属性は未定義のままである．

1Ø	11
11	11

e．もし記憶域参照によって規定される「名前」属性をもつ節が存在するならば，Vはその「値」属性の中に入り，「D」属性に以下の値が入る．

もしこの値なら	⟶	この値になる
Ø		1
1		1
1Ø		11
11		11

前後参照

4.1 「名前付きシステム記憶域」

5.1 「名前付きパーティション記憶域」

5.2 「略式グローバル指標」

12.4 記憶域参照および節指定子へのマッピング

例

1. SET␣^X=Y

   Yと名付けられた階層1の「パーティション」節の「値」属性の内容(すなわち“添字をもたないローカル変数Yの値”)が，^Xと名付けられた階層1の「システム」節（すなわち“添字をもたないグローバル変数”）の「値」属性の中に入れられる．

2. S␣^A(1, 2)=B(3, 4)

   添字付き変数に対しても同様．

3. S␣^A(1, 2)=X, B(4)=7, ^C=“TEST”, X=^D

   SET の複数引数

4. S␣^(2)=^C(X, 2)+1

   =の右の式は=の左の変数名が決定される前に評価される

のでこの場合の略式グローバル参照 ^(2) は ^C(X, 2) を意味する．もしこの行が

S␣^C(X, 2)=^(2)+1

であったとすれば，略式グローバル参照の意味は，SET の実行される前の「略式グローバル指標」の状態に依存する．

5. S␣(A, B, C)=X

これは“複数 SET”である．括弧の中の引数はすべて同じ値を与えられる．

6. S␣(A, B, C, D)=0, ^A(3)=W, (^A(3, 4), ^B(5))=I

“複数 SET”と“単数 SET”を1命令の中で混用できる．

7. S␣^A(5)=10, B(^(3))=^C(4)

＝の左に添字が存在すると，それらは右側の式以前に評価される．ゆえにこの行は次のものと同じである．

S␣^A(5)=10, B(^A(3))=^C(4)

8. S␣^A(3)=10, ^(^(3))=^C(4)

この行は次のように実行される．

a．^A(3) に 10 が与えられる．

b．添字 ^(3) は ^A(3) であるが 10 として評価される．

c．^C(4) が評価される．

d．この値が ^(10) に与えられる．^(10) は ^C (10) のことである．

従ってこの行は次と同じである．

S␣^A(3)=10, ^C(^A(3))=^C(4)

結局，順序は常に次のとおりである．

a．最初に左辺の添字

b．次に右辺の式

c．次に左辺の名前

9. S␣I=3, (I, A(I))=4

“複数 SET” のときでも，左辺の添字は値の割付けがなされる前に評価される．しかも引数の各々は，次の引数に移る前に処理されている．ゆえにこの行は次のものと同じになる．

S␣I=3, (I, A(3))=4

そしてＩは最後には値４をもって終わる．

10. S:X=3␣X=4.5

SET 命令は後付け条件をとりうる．この例では，X=3 であれば，X に 4.5 が与えられる．

11. S␣X=$S(X=3 : 4.5, 1 : X)

$SELECT 関数は SET 命令の右側の式で使うと便利である．この行は前の例と同じ効果をもつ．

12. S␣X=X=3*1.5+X

評価されるべき式が非常に入り組んでいることがある．この例では式 X=3*1.5+X が評価され，その結果が変数 X に与えられる．この行は前の２つの例と同じ効果をもつ[訳注1)]．

13. S␣@X

S␣@A=B+C

S␣A=@B+C

S␣A=B+@(C_D)

SET 命令でしばしば間接指定が用いられる．

訳注 1) 1. X の値が３のとき，X=3 は真であり評価は１，それに 1.5 を掛けて X(=3) を加えるのでこの式の値は 4.5 である．

2. X の値が３でないとき，X=3 は偽であり評価は 0，それに 1.5 を掛けても 0．さらに X を加えるのでこの式の値は X である．従ってこの場合の SET 命令は，

S␣X=X

と同義であって X の値は変わらない．

8.2.16 USE

構文規則

USE ␣ 装置指定式〔:装置パラメータ〕

装置指定式 ::= 式

装置パラメータ ::= | 式 / (式〔:〔式〕〕...) |

詳　解

1. 命令語はＵの１文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができる.
5. 引数の間接指定は許される.

実　行

命令引数中の装置指定式は１つの入出力“装置”を示す. 入出力“装置”は連続した文字の出力源であるかまたは入力口である. この式の値は5.4で解説したようにパーティションの「現用装置指定子」の装置指定子として使われることになる.

USEの目的は「現用装置 指定子」の中に 装置指定式 の値を入れることによって, 目標とする装置を現用装置にすることである. 実行は次のように進行する.

1. 「現用装置指定子」の内容が１つの装置を 指定する ときは, そのときに限りCXとCYの値がその装置に特有のX, Y レジスタに入れられる.
2. 以下の２つの内のどちらか１つが成り立つ.
   a. もし装置指定子の値が１つの装置を指定しそしてもし同一の装置を指定する指定子がパーティションの「Open リスト」内に存在するなら, 装置指定式の値は「現用装置指定子」の中に入れられ, この装置に特

異的な X と Y の値がそれぞれ CX, CY に入れられ，そして OPEN (8.2.12) の解説でみたように装置パラメータが記録される．(解釈：USE 命令によって指定される装置はそれを実行する処理過程の現用装置となる．)

b．以上のようでない場合は何か間違った状態が存在する．

「現用装置指定子」の値は式子として特殊変数 $IO を実行することによって取り出すことができる．

<u>前後参照</u>

4.3　装置指定子

5.4　「現用装置指定子」

8.2.12　装置パラメータ

第9章　特殊変数　$IO

<u>例</u>

1. USE␣3

　　装置 3 は READ と WRITE のデータの通路として指定される．

2. U␣3␣SET␣X=$I

　　X は値 3 をとる．

3. U␣X:Y

　　ある種の言語装備では Y が装置パラメータを指定するのに使われる．

4. U:X'=Ø␣X

　　USE 命令は後付け条件をとることができる．この例は装置 Ø を現用装置とすることを禁じている．

5. U␣@X

　　引数の間接指定が許される．

8.2.17 VIEW

構文規則

VIEW 言語装備者が指定する引数の構文規則

詳 解

VIEW は装備特有の命令であり，その他の方法では取り出すことのできないデータを取り出そうとする言語装備者に役立つ．それが1個または数個の引数をとれるかどうか，そして引数の構文規則が何であるかは，言語装備者によって指定される．それぞれの言語装備は VIEW にどのような解釈法を与えるかを問わず，VIEW 命令を認めて受け入れねばならない．

〔Port: 移送基準に従ったルーチン内では VIEW 命令を用いるべきでない．
:Port〕

8.2.18 WRITE

構文規則

WRITE ␣ | 書式 / 式 / *整数式 |

詳　解

1. 命令語はWの1文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができない.
5. 引数の間接指定は許される.

実　行

WRITE の引数の実行は，データを出力し，そして/または現用装置への情報を制御する．各引数形はそれぞれ異なった一連の規則に従って WRITE の実行を統御する.

WRITE ␣ 書式 は，5.5 と 8.1.4.1 で解説した CX と CY の値を変化させて書式制御情報を現用装置に与えることになる．もし現用装置が出力を受け付けない場合でも CX と CY への効果はそこで解説したとおりであり，そして出力動作は実行されない.

WRITE ␣ 式 は一度に1つの文字ずつ左から右の順で式の値を出力する．それぞれの文字の装置での効果は ASCII 標準と慣例で決められている．出力される各々の文字は 5.5 で解説したように CX と CY の値に影響を及ぼす．もし現用装置が出力を受け入れない場合，CX と CY への効果は 5.5 で解説したとおりであり，そして出力動作は実行されない.

WRITE ␣ *整数式 は1文字の WRITE である．この解釈は多分に装置依

存的であり，それについては言語装備者が定義する．以下に可能性のある解釈法を示す．

1. 装置命令．例えばテープ ユニットを巻きもどせといったもの．
2. 数値による装置パラメータ．例えば ディスク アームの位置 またはプロッタの座標の絶対値など．
3. 整数式の値を ASCII 10 進コードとみなし，それに対応する ASCII 文字を出力する．

言語装備者はこの形式の WRITE の CX と CY への効果を定義する．

時間制限の効果を除いては，READ と WRITE の同一データ を 移送する際の副次的効果は全く同じになるように企図される．

<u>前後参照</u>

5.4　現用装置
5.5　CX と CY
8.1.4.1　<u>書式</u>
8.2.14　READ

<u>例</u>

1. WRITE␣X
2. W␣!, "Line feed", #, "Form feed", ?20, "TAB", A(3)

   WRITE の引数は混合した引数形をとりうる．
3. W␣*7, *7, *97

   前記の解釈法 3 により，この命令は 2 度ベルを鳴らし，その後 "a" を出力する．
4. W␣X, *Y, *Z, !!, "TEST", A=B

   演算子を含む式を引数として使うことができる．
5. W:X>10␣$J(X, 7, 2)

   WRITE 命令は後付け条件をとることができる．$JUSTIFY 関数はしばしば WRITE の引数の中で使われる．

6. W ␣ @X

W ␣ Z, @X, Y

WRITE の引数の中で間接指定を使うことができる.

8.2.19　XECUTE

構文規則

XECUTE␣式

詳　解

1. 命令語は X の 1 文字に省略できる.
2. 命令語は後付け条件をとることができる.
3. 引数は並記できる.
4. 引数は後付け条件をとることができる.
5. 引数の間接指定は許される.

実　行

XECUTE の引数の実行は，6.2の6. のシステム モデルで充分解説されている. これは式の値をそのつづりとし1sではじまり eol で終わる1行のサブルーチンを呼び出す効果がある. サブルーチンの完了時，またはサブルーチンから脱け出た直後，制御は自動的に XECUTE の引数に続く命令にもどる. サブルーチン内での $TEXT や入口参照の判断に必要な文脈上のルーチン体とはこの XECUTE を含んだルーチン体をいう.

前後参照

6.2 の 6. XECUTE の実行

8.2.13　QUIT

例

1. XECUTE␣A

   A が値 "S␣X=1" をとると仮定すれば，X は値1をとる.

2. X␣"S␣X=B␣Q:X<3␣␣W␣Y"␣W␣"OUT"

   この例で引数の値の中にある QUIT はサブルーチンの実行を終わらせることができる. "OUT" は常に書かれる.

3. X␣"G␣A"␣W␣"OUT"

"OUT" は絶対に書かれない.

4. X ␣ A, B—" ␣ D ␣ 3 ␣ G ␣ "—C, D

XECUTE は複数の引数をもつことができる. また命令と引数をいっしょに連結する連結演算を行ないうることに注意.

5. X : Y=1 ␣ A—" ␣ X ␣ B, C", Y

後付け条件をもつ XECUTE. XECUTE の入れ子に注意.

6. X ␣ @Y, "I ␣ @A ␣ G ␣ "—@D

2つのレベルで間接指定をもった例.

8.2.20　Z

構文規則

Z〔言語装備者が指定する 命令語の残りの 部分のつづり〕 言語装備者が指定する引数の構文規則

詳　解

標準にない命令を望む言語装備者は，命令語をZで始まるつづりにすることを標準によって要求される.

〔Port: 移送基準に従ったルーチンは Z 命令をもつべきでない.　:Port〕

第 9 章
特 殊 変 数

特殊変数とは規定された大文字のアルファベット名の頭に $ をつけたもので，式子として実行されると「システム記憶域」あるいは「パーティション記憶域」の中からその値を取り出しうるが，他の方法ではその値を明確に取り出すことができないものを言う．次のような特殊変数が定義されている．

$HOROLOG　「時計レジスター」の内容

$IO　「現用装置指定子」の内容

$JOB　このパーティションに対応する「作業番号 P-ベクトル」要素の内容

$STORAGE　まだルーチンおよび「名前付きパーティション」として使用できるスペースの残量

$TEST　「テスト スイッチ」の内容

$X　CX の内容

$Y　CY の内容

$Z〔言語装備者の定義したつづり〕: 言語装備者の定義による内容

特殊変数は $Z を除いて 2 文字に略して用いうる．すなわち $ と名前の第 1 番目の文字である．

特殊変数は構文規則上次のように定義されている．

特殊変数 ::= | $ H〔OROLOG〕
$ I〔O〕
$ J〔OB〕
$ S〔TORAGE〕
$ T〔EST〕
$ X
$ Y
$ Z言語装備者指定 |

9.1 $HOROLOG

名 前　　$HOROLOG

省略形　　$H

値　　実行時の「時計レジスター」の内容

前後参照　　4.5 「時計レジスター」

例

1. 次のコードは $H を 11：15：01␣AM のような読みやすい時間の形式に変換する.

```
S␣TIME=$P($H, ",", 2), SECONDS=TIME#60
S␣X=TIME\60, HOURS=X\60, MINUTES=X#60
S␣SUFFIX=$P("AM, PM", ",", HOURS\12+1)
W␣!, HOURS+11#12+1, ":", MINUTES, ":", SECONDS, "␣",
    SUFFIX
```

2. 次のコードは $H を MM／DD／YYYY 形式の日付に変換する.

```
S␣H=$H>21608+$H+1460, LEAPYRS=H\1461,
    Y=H#1461訳注1)
S␣YEAR=LEAPYRS*4+1837+(Y\365), DAY=Y#365+1
S␣MO=1␣I␣Y=1460␣S␣DAY=366, YEAR=YEAR-1
F␣I=31, YEAR#4=0+28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
    31␣Q:DAY'>I␣␣S␣DAY=DAY-I, MO=MO+1
W␣MO\10, MO#10, "/", DAY\10, DAY#10, "/", YEAR
```

訳注 1) LEAPYRS; Leap Years（うるう年）をちぢめた変数名.

9.2 $IO

名　前　　$IO

省略形　　$I

値　　「現用装置指定子」の内容

前後参照　5.4「現用装置指定子」

例

1. USE␣3␣S␣X=$I

   X は値 3 を与えられる.

2. S␣^A($I, 1)=X

   データをそのときに使用している装置の番号のもとに一括するファイルを作ることができる.

9.3 $JOB

名 前　　$JOB

省略形　　$J

値　　このパーティションに対応する「作業番号 P-ベクトル」要素の内容

前後参照　　4.4 「作業番号 P-ベクトル」

例

1. S␣^A($J, 2)=Y

   データを作業番号のもとに一括するファイルを作ることができる.

2. WRITE␣#, DATE, "␣", $J

   出力の源を識別する方法.

9.4 $STORAGE

名　前　　$STORAGE

省略形　　$S

値　　$Sの値は，それがどうしても幾分か言語装備に依存しているため，標準による正確な定義をすることができない．$Sはその実行時にあとどれだけパーティション スペースを拡張することをシステムが許容するかという字数を量る．言語装備者が用意するもっと詳しい公式がない場合でも，次のようなモデルを「移送基準」とする．パーティション スペース PS は次の合計として算定される．

PS=RS+LVS+TRS

RS（ルーチンのサイズ）は「ルーチン スタック」の最上部のルーチン体の全行が占める文字数である．（そこでは ls は1文字とみなし，eol は2文字とみなす．）

LVS（ローカル変数の記憶域）は「名前付きパーティション記憶域」の全節の全「値」属性の占める全字数の総計（定義されていない値は空とみなされる）に，これらの節の「名前」属性の占める全データ文字（¢を除く）を足したものに，更にこれらの節数の2倍の数を足し，更にこれらの節で直接上位/直接下位の関係にある組数を2倍にした数を足したものである．

TRS（一時的な結果の記憶域）は現在の行の実行から生じ，$S の実行時に存在する一時的データ値の文字数である．これら一時的データ値には次のものがある．

1. 間接指定の結果として，行バッファに加えられ，そのままとなっている文字．

2. $S を含んだ式の評価中，二項演算子の右側の演算対象が括弧に入っており，その括弧内の評価が終わっていないときの中間結果.
3. $S が SET 命令の中にある場合＝の左側でかつ$S の左側にある記憶域参照の節指定子の中のデータ文字.

〔Port: 移送基準に従ったルーチンでは実行中にいかなる場合でも PS が 4000 を越えることがあってはならない. :Port〕

TAS を$S の実行のときに量られた“利用可能な全スペース”の値とすると

$S=TAS−PS

しかし時に TAS は$S が実行されるパーティション外の活動に影響される関数でもありうる.

前後参照　5.1「名前付きパーティション記憶域」(4.1も参照)
　6.2 の 8. 間接指定
　第7章 行の構造
　8.2.15 SET の実行順

例

1. I␣$S<5Ø␣G␣^OVERFLOW

　$S がある閾値以下であるかを試しオーバーフローに備える.

2. F␣I=1:1:1ØØ␣DO␣3␣I␣$S<1ØØ␣D␣^FILE

9.5　$TEST

名 前　　$TEST

省略形　　$T

値　「テスト スイッチ」の内容（0または1）.「テスト スイッチ」は $T, ELSE, あるいは引数のない IF の実行時には定義されていなければならない.

前後参照　5.3「テスト スイッチ」
8.1.4.2　時間制限
8.2.4　ELSE
8.2.9　IF

例

1. I␣X=3␣S␣Y=A
W:$T␣Y␣S␣Y=B
このコードは以下に相当する.
I␣X=3␣S␣Y=A
I␣␣W␣Y
S␣Y=B
2. R␣X:100␣G␣A:'$T␣DO␣B
この例で $T は制限時間内に入力が明確に終結したかどうかを調べるのに使われる. もし入力が終結しない場合（すなわち入力文字列が未完成であるとき）実行は A から続く.
3. O␣4:10␣G␣A:'$T␣U␣4␣W␣X
これは例2と似ており, OPEN が不成功に終わった場合は A に移る.

9.6 $ X

<u>名 前</u>　$ X

<u>省略形</u>　$ X

<u>値</u>　CX の内容

<u>前後参照</u>　5.5 CX

<u>例</u>

1. W␣!, "123"␣S␣Y= $ X

   Y は値 3 を与えられる.

2. W␣!␣S␣Y= $ X

   Y は値 Ø を与えられる.

3. W: $ X>72␣!

   このコードは $ X が次の出力文字が欄 72 の右に出ることを示すなら復帰改行することを表わす.

9.7　$ Y

名　前　　$ Y

省略形　　$ Y

値　　CY の内容

前後参照　　5.5　CY

例

1. W␣#␣S␣Y＝$ Y

   Y は値 0 が与えられる.

2. W␣#!!!␣S␣Y＝$ Y

   Y は値 3 を与えられる.

3. W：$ Y>56␣#

   その頁が既に一杯であればここで新しい頁にとべ.

9.8 ＄Z

<u>名　前</u>　　＄Z〔言語装備者によって指定される〕

<u>省略形</u>　　＄Z〔言語装備者によって指定される〕

<u>値</u>　　言語装備者によって指定される．標準では言語装備者が定義したすべての特殊変数が＄Zで始まることを要求している．
〔Port: 移送基準に従ったルーチンは特殊変数＄Zを含んではならない．
:Port〕

第 10 章
関　　　数

10.1 概　　要

MUMPS 関数はどれも1個またはそれ以上の関数引数をもつ. そしてこれらが各々式子として実行されるときは, データ文字列としての値を生ずる. 各関数の独特な構文規則は別々に規定されているが, すべての関数に共通な次のような一般的な構文規則がある.

1. 関数とは関数名とそのあとに続く括弧に囲まれた引数並記から成る.
2. 関数名とは規定の大文字のアルファベット名の前に "$" をつけたものである.
3. 複数の関数引数がある場合, となりあった引数はコンマによって分けられる.

定義されている関数名は次のものである.

$ASCII　　文字列の中から1文字を選択し, その ASCII コードを整数で返す.

$CHAR　　整数の並記を ASCII コードのそれらの整数に対応する文字の列に変える.

$DATA　　指定された「名前付き システム 記憶域」ないし「名前付きパーティション記憶域」の節の「D」属性を返す.

$EXTRACT　　文字列の中から位置番号によって抽出した文字や副文字列を返す.

$FIND　　文字列の中の，指定した副文字列の終わりの位置を示す整数を返す．

$JUSTIFY　　指定された長さの文字領域内で右よせした形で式の値を返す．

$LENGTH　　1つの文字列の長さを出す．

$NEXT　　指定された「名前付きシステム記憶域」ないし「名前付きパーティション記憶域」の節の次の兄弟節（最下位レベルの整数添字について，それより大きい添字の節）を見付け出す．

$PIECE　　1つの文字列内である指定された副文字列2つの間の文字列を返す．

$RANDOM　　指定された範囲の偽似乱数を返す．

$SELECT　　数個の式の並記のうちから1個の値を対応する真偽値式の値にしたがって選び出してくる．

$TEXT　　「ルーチン スタック」の最上部のルーチン体の指定された行の文内容を返す．

$VIEW　　言語装備に特異的なデータを得るために利用できる言語装備者によって定義された関数．

$Z〔言語装備者による定義〕　言語装備者によって定義される．

関数名は $Z を除いて $ と名前の頭文字の2文字に省略できる．

構文規則上すべての関数は次のように定義される．

| $A〔SCII〕（式〔，整数式〕）
| $C〔HAR〕（整数式〔，整数式〕...）

関数 ::=

$ D〔ATA〕(記憶域参照)
$ E〔XTRACT〕(式, 整数式〔, 整数式〕)
$ F〔IND〕(式, 式〔, 整数式〕)
$ J〔USTIFY〕(式, 整数式)
$ J〔USTIFY〕(数式, 整数式, 整数式)
$ L〔ENGTH〕(式)
$ N〔EXT〕(記憶域参照)
$ P〔IECE〕(式, 式, 整数式〔, 整数式〕)
$ R〔ANDOM〕(整数式)
$ S〔ELECT〕(真偽値式 : 式〔, 真偽値式 : 式〕...)
$ T〔EXT〕(+整数式)
$ T〔EXT〕(行参照)
〔$ V〔IEW〕(言語装備者によって指定された引数)〕
〔$ Z言語装備者によって 指定された 名前(言語装備者
によって指定される引数)〕

10.1.1 位置番号 (Position Number) の定義

空でない文字列 S が n 文字をもつとすると，それぞれの文字は閉区間〔1, n〕内[訳注1)]で独自の「位置番号」(Position Number) が与えられる．S の一番左の文字は位置番号 1 をもち，S の一番右の文字は位置番号 n をもつ．そしてその中間の文字には左から右の順に中間の整数が与えられている．

逆に，位置番号を表わす整数値 p から S の中の文字への対応 $E(S, p) は，S が文字を含まない文字列[訳注2)]か 1 文字の文字列であっても次のように定義される．

1. もし位置番号 p が 1 よりも小さいか，n より大きい場合 $E(S, p) の値は空文字列となる．(これは S が空のときにいかなる p の値に対しても同様である．)
2. もし位置番号 p が区間〔1, n〕の中にあれば，$E(S, p) の値は位置番号 p をもつ S の 1 文字からなる文字列である．

以上の位置番号から文字への還元が，$EXTRACT(S, p) という 2 引数形式の関数の定義である．

訳注 1) 閉区間〔1, n〕とは 1 以上 n 以下の数値のこと．1 も n も含まれる．

訳注 2) 空文字列のこと．

10.2　関数の定義

10.2.1　$ASCII

<u>名　前</u>　　　$ASCII

<u>省略形</u>　　　$A

<u>構文規則</u>　　$A〔SCII〕(<u>式1</u>〔, <u>整数式2</u>〕)

<u>値</u>　　<u>整数式2</u>がないとき欠損代用値 (default value) 1 が使われる.

$ASCII は $E (<u>式1</u>, <u>整数式2</u>) で返される文字に対応する ASCII 10進コードの整数値を返す. もし, $EXTRACT の値が空であれば $ASCII の値は −1 である.

<u>前後参照</u>　　10.1　$E

附録 A　ASCII 文字表

<u>*例*</u>

<u>第1引数の文字列の値</u>	<u>関数の値</u>
SET␣X="ABCDE"	$A(X)=65
	$A(X, 1)=65
	$A(X, 2)=66
	$A(X, 3)=67
SET␣Y=4	$A(X, Y)=68
SET␣X=""(空文字列)	$A(X)=−1
	$A(X, n)=−1 (すべての n に対して)
SET␣X="AB"	$A(X, 0)=−1
	$A(X, 3)=−1
	$A(X, −7)=−1
	$A(X, 1.92)=65 (整数部分が使われる)

10.2.2 $CHAR

名　前　　$CHAR

省略形　　$C

構文規則　　$C〔HAR〕(整数式〔, 整数式〕...)

値　　$CHAR は長さが引数に等しい文字列を返す．各引数の値は閉区間〔0, 127〕[訳注1] 内の整数である．各整数式は，ASCII 10 進コードがその整数式の値に等しい文字に変換され文字列を作る．この文字列の文字の順序はそれに対応する各整数式の順序に等しい．負の値をとる引数は文字に変換されず $CHAR の値の中には入らない．引数が 127 より大きいときは誤りとみなされる．

前後参照　　附録 A　ASCII 文字表

例

引数の値	関数の値
SET ␣ X=65, Y=66, Z="GOB"	$C(X)="A"
	$C(Y)="B"
	$C(X, Y)="AB"
	$C(X, Y, 67)="ABC"
	$C(X, −1, Y)="AB"
	$C(−1)="" (空の文字列)
	$C(0)=ASCIIのNUL 文字
	$C($A(Z, 1), $A(Z, 2), $A(Z, 3))="GOB"

訳注 1)　カナを装備した標準 MUMPS ではこの限りではない．

10.2.3 $DATA

名　前　　$DATA

省略形　　$D

構文規則　　$D〔ATA〕(記憶域参照)

値　　引数によって指定された節が存在するときは，関数の値はその「D」属性の内容である．その節が存在しないときは関数の値は0である．記憶域参照が略式グローバル参照であれば「略式グローバル指標」が定義されていなければならない．記憶域参照がグローバル変数または略式グローバル参照であれば「略式グローバル指標」の内容は変化しうる．

前後参照　　4.1　「D」属性

5.2　「略式グローバル指標」

12.4　記憶域参照

例

1. 「名前付きパーティション記憶域」がその初期状態にあると仮定する．

引数の値	関数の値
Y が定義された値をもたない.	$DATA(Y)=0
SET ␣ Y=100	$DATA(Y)=1
SET ␣ Y="AB"	$D(Y)=1
SET ␣ A(1)="TEST"	$D(A(1))=1
	$D(A)=10
SET ␣ B(1, 2)="SAMPLE"	$D(B(1, 2))=1
	$D(B(1))=10
SET ␣ B(1)="ANOTHER ␣ SAMPLE"	$D(B(1))=11
KILL ␣ B(1, 2)	$D(B(1, 2))=0
	$D(B(1))=11 (KILL はいかなる上位の「D」属性も変えない.)

2. 引数がグローバル変数であり「名前付きシステム記憶域」が SET␣^A(1,2,3)="TEST" が実行されただけでそれ以外は初期の状態にあると仮定する.

操　作	Xの値	「略式グローバル指標」の結果値
SET␣X=$D(^A)	10	定義されていない
SET␣X=$D(^(1))	誤	定義されていない
SET␣X=$D(^A(1))	10	^A
SET␣X=$D(^A(99))	0	変化していない
SET␣X=$D(^(1))	10	変化していない
SET␣X=$D(^(1,2))	10	^A¢1
SET␣X=$D(^(2))	10	変化していない
SET␣X=$D(^(2,3))	1	^A¢1¢2
SET␣X=$D(^(3))	1	変化していない
SET␣X=$D(^(4))	0	変化していない
SET␣X=$D(^A(1,2))	10	^A¢1
SET␣^A(1,2,3,4)="H", X=$D(^A(1,2,3))	11	^A¢1¢2

10.2.4 $EXTRACT

名　前　　$EXTRACT

省略形　　$E

構文規則　　$E〔XTRACT〕(式1，整数式2〔，整数式3〕)

値　　整数式3がないなら10.1.1に述べられた定義が使われる．整数式3があるときは $EXTRACT の値は，次の連結式によって定義されるような式1の連続した副文字列である．

整数式3の値が整数式2の値より小さくなければ，

$E(S, m, n)=$E(S, m)_$E(S, m+1)_ . . . _

$E(S, n−1)_$E(S, n)　.

整数式3の値が整数式2の値より小さいならば関数の値は空文字列となる．

前後参照　　10.1.1　2引数形式の$E の定義

例

初めの条件	関数の値
SET␣X="ABCDE"	$E(X, 1)="A"
	$E(X, 2)="B"
	$E(X, 1, 2)="AB"
	$E(X, 1, 4)="ABCD"
	$E(X, 1, 100)="ABCDE"
	$E(X, 1. 9)="A"（引数の整数部分が使われる）
	$E(X, 99)=""（空文字列）
	$E(X, −3)=""（空文字列）
	$E(X, 3, 2)=""（空文字列）
SET␣A(1)="2BC"	$E(A(1), A(1))="B"（第2引数の整数評価した値が使われる．）

10.2.5 $ FIND

名 前　　$ FIND

省略形　　$ F

構文規則　　$ F〔IND〕(式1, 式2〔, 整数式3〕)

値　　整数式3がないときや，あってもその値が1より小さいときその値に欠損代用値(default value) 1が当てられる．$ FIND は式2の値が式1に副文字列として含まれているかどうかを調べる．位置番号が整数式3の値である文字から始めて左から右へ調べられる．

整数式3の値が式1の長さよりも大きいときや式2の文字列が見つからないときは $ FIND はゼロになる．これら以外のときは最初（最も左に）見つけられたものが次のように報告される．見つけられた式2の式1中での位置番号が閉区間〔i, j〕内の整数であるならば $ FIND の値は j+1 である．

前後参照　　10.1.1 位置番号

例

初めの条件	関数の値
SET␣X="ABCAX"	$ F(X, "A")=2
	$ F(X, "B")=3
	$ F(X, "Z")=Ø
	$ F(X, "ABC")=4
	$ F("ABC", "ABC")=4
	$ F(X, "A", 1)=2
	$ F(X, "A", 2)=5
	$ F(X, "A", 4)=5

	$F(X, "A", 5)=Ø
	$F(X, "A", 100)=Ø
	$F(X, "")=1
	$F(X, "", 4)=4
	$F(X, "", 10)=Ø
SET␣Y="B"	$F(X, Y)=3
SET␣Z="1.2W"	$F(X, Y, Z)=3

10.2.6 $JUSTIFY

名 前　　$JUSTIFY

省略形　　$J

構文規則　　$J〔USTIFY〕(| 式1, 整数式2 / 数式1, 整数式2, 整数式3 |)

値　　2引数形式の場合はスペースを整数式2の値だけもつ文字領域に式1を右詰めで書く．L1を式1の長さとしN2を整数式2の値とおく．もしN2≤L1 であればけた落としはなく $J の値は式1の値のままである．N2>L1 であれば，$J の値は左側に N2−L1 個のスペースをおき，その右側に式1の値を連結したものとなる．

3引数形式の場合は，N3を整数式3の値とおく．この形式は数式1の値の小数点以下第N3位までをスペースN2個の文字領域内に右詰めに置いた書式編集をする．特にここで数式1を四捨五入して少数点以下第N3位まで求めらた値をRとする．このとき必要ならば0を連ねることもある．(N3=0 の場合Rには小数点をつけない．) この場合 $J の返す値は $J(R, N2) である．

〔Port: N3の負の値は将来 $J の定義を拡大するために保留されている．ゆえに言語装備者もユーザーも共にこれの使用を避けるべきである． :Port〕

前後参照

例

初めの条件	関数の値
SET␣X=12.35	$J(X, 6)="␣12.35"
	$J(X, 5)="12.35"

	$J(X, 4)="12.35"
	$J(X, 3)="12.35"
	$J(X, 7, 4)="12.35ØØ"
	$J(X, 7, 3)="␣12.35Ø"
	$J(X, 7, 2)="␣␣12.35"
	$J(X, 7, 1)="␣␣␣12.4"
	$J(X, 7, Ø)="␣␣␣␣␣12"
SET␣Y=197	$J(Y, 7, 2)="␣197.ØØ"
SET␣Z=5.4	$J(Z, 7, 2)="␣␣␣5.4Ø"
SET␣W=1.487	$J(W, 7, 2)="␣␣␣1.49"
	$J(W, 7, 1)="␣␣␣␣1.5"
	$J(W, 7, Ø)="␣␣␣␣␣␣1"

10.2.7 $LENGTH

名　前　　$LENGTH

省略形　　$L

構文規則　　$L〔ENGTH〕（式）

値　　$LENGTH（式）は式の値の文字数を整数値で返す．式の値が空なら $L はゼロになる．

前後参照

例

初めの条件	関数の値
SET␣X="ABC"	$L(X)=3
SET␣X="123456789"	$L(X)=9
SET␣X=""（空文字列）	$L(X)=0

10.2.8　$ NEXT

名　前　　$ NEXT

省略形　　$ N

構文規則　　$ N〔EXT」(記憶域参照)

値　　添字のついた（レベル2あるいはそれより大きい）引数形式だけが許される．そして最も右の添字の値は，その値が −1 以上の整数でなければならない．引数が略式グローバル参照なら「略式グローバル指標」は定義されていなければならない．

引数が次の形であるとする．

N(s1, s2, . . . , sn−1, sn)

ここで sn は前記のような制約をもっている．$ NEXT は以下のような記憶域参照によって指定された記憶節があるか，そしてあればその性質について知らされる．

N(s1, s2, . . . , sn−1, t),

1. 節は Ø でない「D」属性をもつ．
2. t は sn より大きい整数である．
3. t は 1., 2. の条件を満足させる最小の値である．

このような節が存在するならば $ NEXT の値は t である．このような節が存在しなければ，NEXT の値は −1 である．

前後参照　　4.1　記憶域のレベル

4.1　「D」属性

12.4　記憶域参照

<u>例</u>　「名前付きパーティション記憶域」と「名前付きシステム記憶域」は SET␣^A(1, 2, 3)="TEST", ^A(4)="SAMPLE" の実行の結果を除いて，初期の状態にあると仮定する.

操　作	Xの値	「略式グローバル指標」の結果値
SET␣X=$N(^A)	誤	定義されていない
SET␣X=$N(^A(−1))	1	^A
SET␣X=$N(X(4))	−1	変化なし
SET␣X=$N(^(1))	4	変化なし
SET␣X=$N(^(4))	−1	変化なし
SET␣X=$N(^(4, 3))	−1	変化なし
SET␣X=$N(^A(1, −1))	2	^A¢1
SET␣X=$N(^(2, −1))	3	^A¢1¢2
SET␣X=^(3)	"TEST"	変化なし
SET␣X=$N(^(3))	−1	変化なし
SET␣X=$N(^B(1, 2))	−1	定義されていない

10.2.9 $PIECE

名　前　　$PIECE

省略形　　$P

構文規則　　$P〔IECE〕(式1，式2，整数式3〔，整数式4〕)

値　整数式4がなければ整数式3の値が整数式4の値として使われる．S1を式1の値，S2を式2の値，N3を整数式3の値，N4を整数式4の値と仮定する．

$PIECE(S1, S2, N3, N4) は両端が S2, N3, N4 によって定められるS1の副文字列を求め，それを $P の値として返す．典型としては，求める副文字列はS1の左端から数えて N3−1 番目のS2に左が接し（しかもそれを含んでいない）右はS1の左はしから数えてN4番目のS2に接している（しかもそれを含まない）．空の副文字列はどの番号のところにも見当たるはずであるから，S2が空ならば $P は空文字列を返す．

次のどの条件でも $PIECE は空文字列を返す．

1. N3>N4
2. N4<1
3. S1に含まれるS2の個数が N3−1 個より少ない場合
4. S2が空文字列のとき

上の条件のどれにもあてはまらなければ，$PIECE の値はS1内のN3−1番目のS2とN4番目のS2に接する副文字列である．

S1内のS2の出現はS1の左端から数えられ重複しないものとされる．このようにすれば“ABABABABAB”には“ABAB”が2回出現

する.

N3が1以下のときは，取り出された副文字列はS1の左端を含む. N4がS1内でのS2の出現回数より大きいならば，取り出された副文字列はS1の右端を含む.

前後参照

例

初めの条件	関数の値
SET␣X="ABC*DEF"	$P(X, "*", 1)="ABC"
	$P(X, "*", 2)="DEF"
	$P(X, "*", 3)="" (空文字列)
	$P(X, "B", 1)="A"
SET␣Y="B"	$P(X, Y, 1)="A"
	$P(X, Y, 2)="C*DEF"
	$P(X, "/", 1)="ABC*DEF"
SET␣Z="A. B. C. D"	$P(Z, " . ", 1)="A"
	$P(Z, " . ", 2, 3)="B. C"
	$P(Z, " . ", 1, 100)="A. B. C. D"
	$P(Z, " . ", 3, 2)="" (空文字列)
	$P(Z, " ", 1, 100)=""

10.2.10　$RANDOM

<u>名　前</u>　　$RANDOM

<u>省略形</u>　　$R

<u>構文規則</u>　　$R　ANDOM〕(<u>整数式</u>)

<u>値</u>　　Nを引数の値と仮定する．Nは正でなければならない．$R(N)は閉区間〔0, N−1〕内で一様に分布する乱数あるいは擬似乱数を返す．

<u>前後参照</u>

<u>例</u>

1. SET␣X=$RANDOM(N)

   Xは0以上N−1以下の1つの整数値をランダムに与えられる．

2. SET␣Y=$R(2)

   Yは値0か1をランダムに与えられる．

3. SET␣Y=$R(1)

   Yは0である．

10.2.11 $SELECT

名 前　　$SELECT

省略形　　$S

構文規則　　$S〔ELECT〕(真偽値式：式〔，真偽値式：式〕...)

値　　$SELECT の各引数は，コロン：によって分けられた真偽値式：式という順序のペアである．関数は何個の引数をもってもよい．しかし少なくとも1個の真偽値式が値1（真）をもたねばならない．

まず，真偽値式のペアとなっている式は評価しないで，各真偽値式が一度に1つずつ左から右に向かって評価される．この評価は真偽値式が1となる最初の引数のところで止る．他の引数は除外して，この引数内の式の値が求められ，これが$SELECT の値となる．他の式は値を定義される必要がないことに注意．

前後参照

例

初めの条件	関数の値
SET␣X=1	$S(X=1：8, 2=3：0)=8
	$S(X=1：8, 2=2：0)=8
	$S(X=2：8, 2=2：0)=0
	$S(X=2：8, 2=3：0)=誤； 1つの引数はどうしても「真」の真偽値をとらねばならない．
SET␣Y="B"	$S(X=3：8, Y="B"："HELLO", X=1：13)="HELLO"
	$S(X=Y：B(1), Y="A", ^(2, 3), 1：3)=3 この例では「略式グローバル指標」を用いたり

	変化させたりしないことに注意.
SET␣^A(1)="TEST"	$S(X=1 : ^A(1), Y : S)="TEST" 「略式グローバル指標」は ^A に変えられる.

10.2.12 $TEXT

名　前　　$TEXT

省略形　　$T

構文規則　　$T〔EXT〕(| +整数式 / 行参照 |)

値　　引数は「パーティション スタック」の最上部にあるルーチン体の1行を表わす．この表示は次の方法のどちらかでなされる．

1. 引数が+整数式をもつならば，その値はこれをNで表わすとゼロより大きくなければならない．表示される行はルーチン体のN番目の行である．（第1行はN=1にあたる．）
2. 引数が行参照形式をもっているなら，行は第7章で述べられた行参照法によって表わされる．

ルーチン体の行頭にないラベルを含んでいる行参照，あるいは整数式の値が大きすぎるために，$TEXTの引数が1つの行を指定しないとき，$TEXTの値は空文字列となる．このようなことがないときは次の点を除いて$TEXTの値は指定した行の値となる．

その点とはすなわち1sが1つのスペース（SP）で置き換えられ，かつeolが抹消されることである．

前後参照　　第6章「パーテイション スタック」

第7章　ルーチン体

第7章　行参照

例

次のルーチン体が「パーティション スタック」の最上部にあると仮定する．

```
ABC ;SAMPLE
        SET␣X=3␣WRITE␣Y
        READ␣X
Z       WRITE␣!,X
```

このとき

$T(ABC)="ABC␣;SAMPLE"

$T(Z)="Z␣WRITE␣!,X"

$T(ABC+1)"␣SET␣X=3␣WRITE␣Y"

$T(ABC+2)="␣READ␣X"

$T(+1)="ABC␣;SAMPLE"

$T(+3)="␣READ␣X"

$T(Z+1)=" "

$T(@A+2)="␣READ␣X"

(A="ABC" が与えられている)

$T(ABC+K)="␣READ␣X"

(K=2 が与えられている)

$T(+B)="␣READ␣X"

(B=3 が与えられている)

次の 2 つの行はこれらを含むルーチン体全体を書き出す.

```
F␣I=1:1␣S␣X=$T(+I)␣Q:X=""␣␣W␣X,!
W␣#
```

10.2.13 $VIEW

名　前　　$VIEW

省略形　　$V

構文規則　　$V〔IEW〕（言語装備者によって指定された引数構文）

値　　$VIEWはそれ以外の方法では利用できないある種のデータをプログラムに利用させたい言語装備者によって用意される言語装備時の関数である．どの言語装備も，いかなる解釈によろうとも，$VIEW関数を認め受け入れなければならない．

〔Port: 移送基準に従ったルーチンは $VIEW 関数を含んではいけない． :Port〕

10.2.14　$Z

名　前　　　$Z 言語装備者によって残りは指定される

省略形　　　$Z 言語装備者によって指定された名前の省略形

構文規則　　$Z 言語装備者によって指定（言語装備者によって指定された引数構文）

値　　標準にない関数を用意したい言語装備者は，その関数名を$Z ではじめてつづることを標準によって要求されている．

〔Port: 移送基準に従ったルーチンは いかなる $Z 関数をも用いてはならない．　:Port〕

第 11 章

式

11.1　式を支配する一般的法則

11.1.1　式の構文規則

式は，命令の一部を形成し，実行されたときに値を生ずる．副文字列である．式の実行は添字，関数の引数，および記憶節への割付け値を得る主要な手段となる．

式は移送性（第７章参照）のために行の長さを指定されるというようなある種の制限はあるが，適宜複雑なものになる．しかしながら典型として式はただ２つの要素からなる．すなわち式子と二項演算子とである．式子１つ（第12章参照）は，式の最も単純な形である．それは行の副文字列で，値を持ちうる最小単位であり，それが集ってさらに複雑な式が合成される．二項演算子は式子どうしをつなぎ，さらに複雑な式をつくる．

MUMPS ルーチンの実行中は，ほぼ１つおきに行の文字群を左から右へ走査する過程で式が実行されている．この基本的な進行の構造を式の構文規則定義の上で明確にしてみると次のようになる．

式　::=　| 式子
　　　　　| 式　式尾

上の循環形の公式の右辺はより単純に反復形の公式で表わせば，構文規則上次のようになる．

<u>式子</u> 〔<u>式尾</u>〕...

しかし循環形の公式は，計算の模型を示す上ではより厳密な表現である．式尾はどれも2つの部分からなり，左に二項演算子をもち，右に式子かパターン訳注1)を右にもつ．

訳注 1)　パターン；11.2.4.3 参照．

11.1.2 MUMPS 値

V をすべての MUMPS 値を含む全体集合とすると，それは7-ビットの ASCII 文字からなるすべての文字列の集合である．(これら128個の ASCII 文字の表は付録 A を参照のこと．) この全体集合は空文字列を含む．記憶容量のような実際上の制限を除いて，この全体集合から1つの値をとってみると，その値を作りだすための MUMPS ルーチンを書くことができる．

〔Port: 移送基準をふまえた MUMPS ルーチンによって作りうるすべての値を含む全体集合は，長さが255以下のすべての ASCII 文字列であって，長さが255を越える文字列は含まない． :Port〕

V には4つの重要な部分集合がある．すなわち指数表記法で書かれる数値である部分集合 Ve，その部分集合である指数なしの数値 Vn，その部分集合である整数値 Vi，そしてその部分集合である真偽値 Vt である．これらの部分集合は，順に前のものが後のものを包んでいる[訳注1]．これらの部分集合は，その要素である文字列の構文規則によって定義される．部分集合の構成要素と構文規則との関係は次の表のとおりである．

MUMPS 値の部分集合における文字列	構文規則	定義場所
Ve	指数値	11.1.3
Vn	数　値	11.1.2
Vi	整数値	11.1.2
Vt	真偽値	11.1.2

数値，整数値および真偽値の構文規則の定義は次のとおりである．

真偽値 ::= | Ø |
　　　　　 | 1 |

訳注 1) $V \supset Ve \supset Vn \supset Vi \supset Vt$.

0でない数字　::=　| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

数字　::=　| 0 | 0でない数字 |

整数値　::=　| 0 | 〔−〕0でない数字〔数字〕. . . |

〔Port: 移送基準をふまえたMUMPSルーチンでは実行によって生ずる整数値のけた数は9を越えてはならない.　:Port〕

小数値　::=　.〔数字〕. . . 0でない数字

（注：1つのピリオドは“.”で示してあるが，3つのピリオド群は〔数字〕という構文要素を任意個繰り返すことを示す.）

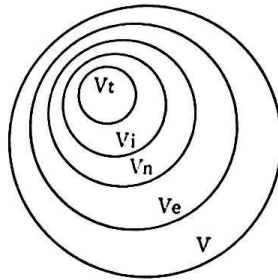
数値　::=　| 0 | 〔−〕0でない数字〔数字〕. . .〔小数値〕 | 〔−〕小数値 |

各々の整数値あるいは数値は，それぞれ独自の実数を意味し，その数は現実的なものである. 真偽値0は「偽」を意味し1は「真」を意味する.

〔Port: 移送基準をふまえたMUMPSルーチンによって生ずる数値はその絶対値が閉区間〔10^{-25}，10^{25}〕に含まれる数または0でなければならない.

:Port〕

11.1.3 解 釈 演 算

MUMPS 値の集合 V, Ve, Vn, Vi および Vt は，次のベン図のような順次包含される集合列を形成する．

すなわちあらゆる真偽値は整数値であり，あらゆる整数値は数値であり，あらゆる数値は指数値であり，あらゆる指数値は MUMPS 値である．多くの演算子は既に定義されているが，それらの定義は V の部分集合ただ1つのものについてであって，V の全体にわたるものではない．いくつかの例を次に示そう．

1. 算術演算子は数値に関して定義されている．
2. ブール演算子は真偽値に関して定義されている．
3. $CHAR や $EXTRACT のようなある種の関数は，整数値として扱われるいくつかの引数をもつ．

MUMPS 言語の設計上の原則なのであるが[訳注1)]，V の部分集合に対して定義されているどんな演算も，たとえその演算子の属する部分集合の外の引数がきても，実際には誤りとはならないのである．誤りとはせずにむしろ限定された引数を必要とする演算であっても，演算自体が実行される前に，それぞれ必要とするような引数に“解釈”演算を実行してしまうのである．その適当な解釈演算とは全 MUMPS 値を必要とされる部分集合に位置がえすることである（も

訳注 1) 〰〰〰〰線は訳者による．

ちろん正しい構文規則を持った値を変えるものではないが). それによって,演算の定義に用いられた制限にすべての引数値を合わせることを保証している.

関数, 演算子, 命令などの引数で必要となる特定の解釈演算は, その引数の構文形によって決まってくる.

引数の構文形	値が含まれるべき部分集合	引数実行以前になされる解釈演算
式 式　子	V	なし
指数式 指数式子	Ve	指数解釈 Ie
数　式 数式子	Vn	数解釈 In
整数式 整数式子	Vi	整数解釈 Ii
真偽値式 真偽値式子	Vt	真偽値解釈 It

ルーチン自体において, これらの特定な引数形がより一般的な引数形と同一構文形をもっていることに注意すること. すなわち,

指数式子 ::= 式子
数式子 ::= 式子
整数式子 ::= 式子
真偽値式子 ::= 式子

指数式 ::= 式
数　式 ::= 式
整数式 ::= 式
真偽値式 ::= 式

解釈演算とは，値を V の全体からそれが必要とされる部分集合へ置き換えることである．しかしこの解釈演算は既にその必要とされる部分集合内にあるものの値を変えることはない．

これは次のように図式的に表現される．

$$V \xrightarrow{\text{Ie}} \text{Ve}$$

$$V \xrightarrow{\text{In}} \text{Vn}$$

$$V \xrightarrow{\text{Ii}} \text{Vi}$$

$$V \xrightarrow{\text{It}} \text{Vt}$$

一方 MUMPS の算術演算子は，指数値または“科学的”な形[訳注1)]では結果をださないが，数解釈 In は，正確にこの形の引数データを解釈し，それを数の形に転換する．これが，指数値それ自体が，算術演算の定義内に用いられていないにもかかわらず，指数値集合 Ve の概念を導入した理由である．（第12章では式子の数字列形の解釈と関連してデータの指数形がもっと明確に用いられている．）Ve の要素を定義する指数値の構文規則定義は次のとおりである．

数字連鎖 ::= 数字〔数字〕...

仮　数 ::= | 数字連鎖〔．数字連鎖〕 / ．数字連鎖 |

指　数 ::= E〔±〕数字連鎖

指数値 ::= 〔−〕仮数〔指数〕

In, Ii および It の解釈は，その後1つ以上の基本的解釈である Ie, Ien, Ini および Int[訳注2)]に細分化されてゆく．それは次の図式と表で表現される．

訳注 1）たとえば，1230 という数値は 1.23×10^3 と表わすことができるが，コンピュータでは，しばしばこれを 1.23E3 と表わす．この 1.23E3 という表現をここでは“科学的”な形といっている．

訳注 2）Ien：指数-数解釈　Ini：数-整数解釈　Int：数-真偽値解釈

$$
\begin{array}{ccccccc}
V & \xrightarrow{Ie} & Ve & \xrightarrow{Ien} & Vn & \xrightarrow{Ini} & Vi \\
 & & & & \downarrow{\scriptstyle Int} & & \\
 & & & & Vt & &
\end{array}
$$

この解釈を得るには	この連鎖を適用する
In	Ie に続く Ien
Ii	In に続く Ini
It	In に続く Int

11.1.3.1 解釈 Ie

何らかのデータ文字列 S を集合 Ve に含まれる文字列に変換するのは2つの段階によって行われる．第1には前に付いている符号をすべて単純化することである．第2には指数値を満たす最大のヘッドを見いだすことである．

ヘッドの定義

ある文字列の1つの文字のすぐ右側か左側に1つの点を置いてみたとき，その点についてのその文字列のヘッドとは，この点より左側の文字をすべて含み，右側の文字を何も含まない部分文字列をいう．ヘッドは空であることも，文字列すべてであることもありうることに注意．

1. 次の冒頭符号短縮規則を S に対し可能な限り何回でも，どんな順序でもあてはめる．
   - a．もしSが＋Tの型であれば＋を取り除く．
     (略記法：＋T→T)
   - b．−＋T→−T
   - c．−−T→T
2. 次の2つの場合のどちらか適当な方を応用する．
   - a．S の最も左側の文字が“−”でなければ，結果を指数値の構文をもつSの最も長いヘッドとする．
   - b．S が −T の型であれば，結果を指数値の構文をもつ S の最も長いヘッドとする．もし結果の仮数部分が0（または −0）を示すなら，全数値を“0”とおく．

11.1.3.2 解釈 Ien

1. もし引数の仮数部分に“．”がないならば，仮数の右端に“．”を置く．
2. 指数の存在は“E”によって認識される．それがないときは，ステップ3を跳ばす．

3. もしその指数が１つの＋をもつかまたは何も符号がない場合には，その“．”を指数の数値の10進けた数だけ仮数の中を右に移動させ，必要なだけ0を仮数の右につけ加える．もし指数にマイナスの印があるなら，“．”を指数の絶対値の10進けた数だけ仮数の中を左に移動させ，必要なだけ0を仮数の左につけ加える．
4. 仮数の前または後につくゼロと指数を削除する．
5. 最も右側の文字が“．”ならば，それを取り除く．
6. 結果が空かあるいは“−0”ならば，それを“0”とおく．

11.1.3.3 解釈 Ini

1. 小数値の存在は“．”によって認識される．もしこれがあればその小数値を削除する．
2. 結果が空あるいは“−”ならば，それを“0”とおく．

11.1.3.4 解釈 Int

もしその数が“0”でないならば，それを“1”とおく．

11.1.4 式値の計算

式が生じるときの構文規則は，次の2つの形式のうちの1つをとる．

式子

または

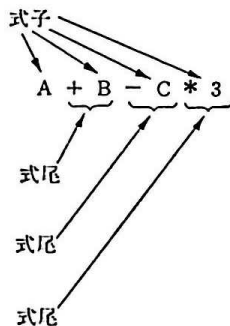
式 式尾

第12章で各式子の値について述べる．式が式子形をとるならば，式の値は式子の値である．これ以外のものでは式は

式 式尾

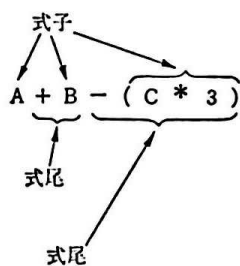
の形をとり，その評価のために次の一般的解説が適用される．

MUMPS では式を左から右に向かって走査する．式尾の右端の文字が丁度走査され終わったときに，式が評価される．従って式そして/または式子の構成要素すべての値が既知となっている．もっとはっきり言えば左側の式の値が既知となり，そして式尾の中のいかなる式子の値も既知となっているということである．

それぞれ式尾が走査されるたびに直ちに評価されることと，走査が左から右に行われることにより，例えば数個の式尾を含む1つの式

は当然，左の方からグループ化されることになり，括弧を使って表わせば，そ

れは次のようになる.

$$(((A+B)-C)*3)$$

もし違った計算順序をとらせたいならば，ある文字の組合せが式子として扱われるように括弧を用いる．すなわち，

11.2 式 尾 の 型

次の式尾の二項演算子のみを考察することによって

式 式尾

すべての式の値が Vt, Vi, Vn あるいは V の分類の どれに含まれるかは予想できるであろう．ゆえにこれを根拠として式尾を次の 4 群に分類することができる．これらの群の各々は結果の値を指名された値の集合に “限定” する．

1. 無限定式尾．この群の唯一の演算子は文字列連結演算子（_）である．
2. 数値限定式尾．この群は，算術演算子 ＋－／*＃ を含む．
3. 整数値限定式尾．この群の唯一の演算子は，整数除算演算子（＼）である．
4. 真偽値限定式尾．この群は関係演算子（＝＜＞〕〔），論理演算子（＆！）およびパターン照合演算子（？）を含む．これらの演算子はすべて，否定接頭子（'）を直前に置いて，真偽値の結果を逆にすることができる．

式尾の構文定義は，次のようになる．

```
                  | | 文字列演算子       |      |
式尾     ::=      | | 算術演算子         | 式子 |
                  | | 〔'〕真偽値演算子  |      |
                  |   〔'〕 ? パターン          |

文字列演算子 ::=  _

                  | ＋ |
                  | － |
算術演算子   ::=  | *  |
                  | ／ |
                  | ＼ |
                  | ＃ |

真偽値演算子 ::=  | 関係演算子 |
                  | 論理演算子 |
```

関係演算子 ::=

| < |
| > |
| = |
| 〔 |
| 〕 |

論理演算子 ::=

| & |
| , |

11.2.1 無限定式尾

演算子　　　_ (文字列連結演算子)

式の構文規則　式1　_　式子2

値の計算　　　この結果生じる式の値は，式1の値の右に式子2の値を連結したものである．

前後参照

例　　X が値 "DE" をとると仮定する．

式	値
"A" _ "B"	AB
"A" _ "B" _ "C"	ABC
"A" _ "1" _ 23	A123
"AB" _ X _ (3*4)	ABDE12
1 _ 2	12
1 _ 2*3	36
1 _ (2*3)	16

11.2.2 数値限定式尾

演算子　　＋（代数の加算）

－（代数の減算）

＊（代数の乗算）

／（代数の除算）

＃（モジュロ）

式の構文規則

	＋	
数式1	－	数式子2
	＊	
	／	
	＃	

値の計算　〔Port: 移送基準をふまえたルーチンはいかなる中間または最終値としても9けた以上の有効数字を必要とする計算を含んではならない.　:Port〕

どちら側の引数にも数解釈 In がとられる.

結果の値はそれが示す実数によって，次のように定義される.

＋は，代数和を生ずる.

－は，代数差を生ずる.

＊は，代数積を生ずる.〔Port: 積の有効数字が9けたを越える場合には，結果の値に一律な定義を与えることはできない.　:Port〕

／は，代数商を生ずる. 商の値は一方の引数が正で，他方の引数が負という場合に限って負となる. ゼロで割ることは認められない.〔Port: 商の有効数字が9けたを越える場合，あるいは商が無限小数を含む場合に

は，結果の値に一律な定義を与えることはできない． :Port〕

♯は，左の引数の右の引数によるモジュロ値を出す．♯は右の引数のゼロでない値に関してのみ次のように定義される．

A♯B＝A－(B*floor (A／B))

ここで floor(X)＝最大整数≦X[訳注1)]

〔Port: 結果における有効数字が9けたを超える場合には，結果の値に一律な定義を与えることはできない． :Port〕

前後参照 11.1.3 数解釈 In

例 A＝0, B＝1, C＝2 とおくと

式	値
2+3	5
2*3+4	1Ø
4+2*3	18
(4+2)*3	18
4+(2*3)	1Ø
4+(2／3)	4.66666667 (近似的)
1+.123456789	1.12345679 (近似的)
A－B－C	－3
A－(B－C)	1
B♯C	1
C♯B	Ø
B+2*C	6
B+(2*C)	5

訳注 1) floor(X)＝(X を越えない最大整数) を意味する．

11.2.3　整数値限定式尾

演算子　　　＼（整数除算）[訳注1)]

式の構文規則　　数式1　＼　数式子2

値の計算　　　結果の値は　数式1　＼　数式子2

の値の整数解釈 Ii である．

〔Port: 結果における有効数字が9けたを越える場合には，結果の値に一律な定義を与えることはできない．　:Port〕

前後参照　　　11.2.2　除算演算子　／

例

式	数値
4+2＼3	2
4+(2＼3)	4

訳注 1)　文字“＼”は ASCII コードでは 92 であるが，JIS コードの 92 は文字“￥”に当てられる．

11.2.4　真偽値限定式尾

演算子　　関係演算子（11.2.4.1参照）

　　=　（文字列合同）

　　<　（代数の　より小さい）

　　>　（代数の　より大きい）

　　〔　（文字列が含む）

　　〕　（文字列が追従する）

　論理演算子（11.2.4.2参照）

　　&　（および，論理積）

　　!　（または，論理和）

　パターン照合演算子（11.2.4.3参照）

　　?

11.2.4.1　関係演算子

式の構文規則

数式1　〔'〕| < | 数式子2
　　　　　　| > |

式1　〔'〕| = | 式子2
　　　　　| 〔 |
　　　　　| 〕 |

値の計算　　　すべての演算子をここで op と表わせば

A　'op　B は '(A　op　B) と同値.

A　op　B はそれの表現する関係が正しければ1の値をとり，そうでなければ0をとる.

不等号の > や < は，これらの引数を数解釈した値を比較する．すなわちこれらの表現する関係は，習慣的な，代数の“greater than（より大きい)”ある

いは “less than（より小さい）” である．

演算子＝は，２つの引数が（いかなる解釈も行わず）合同の文字列であるときに限り，数値１を出す．もしも両引数が数値形であれば，その数表示の独自性によって，数的に同値であるかどうかを調べる＝が満足されることになる．〔Port: ある引数の計算において何かの切り捨てが起きたならば，計算上不正確な結果がでることを考慮しておく．　:Port〕

関係演算子〔は “含む（contains）” と呼ばれる．A〔B は，B が A の中の連続的な副文字列であるときに限り真である．すなわち A〔B は ''$F（A, B）と同じ値をもつ．空文字列は，あらゆる文字列の副文字列である．

関係演算子〕は “追従する（follows）” と呼ばれる．A〕B は，A が慣習的な ASCII 文字表の順序により，辞書編集上の順序で B のあとにくるときに限り真となる．A は次のどれかが真であるときに限り，B に追従すると定義される．

a．B が空で A がそうではない．

b．A も B も空でなく，A の左端の文字が B の左端の文字よりも ASCII コード上数的に大きな値をもつ．

c．A と B が，正の整数 n の長さの全く等しいヘッド[訳注1)]部をもち，A の残りの部分が B の残りの部分に続く．

11.2.4.2　論理演算子

<u>式の構文規則</u>

<u>真偽値式1</u>　〔'〕 | & / ! | <u>真偽値式子2</u>

<u>値の計算</u>　　&と！は，それぞれ “論理積（and）” と “論理和（or）” の名で呼ばれる．

訳注 1）　11.1.3.1 ヘッドの定義参照．

$$\mathrm{A\,!\,B}=\begin{cases}\emptyset & \text{AとB両方が値}\emptyset\text{をとる場合}\\ 1 & \text{それ以外の場合}\end{cases}$$

$$\mathrm{A\,\&\,B}=\begin{cases}1 & \text{A と B の両方が値1をとる場合}\\ \emptyset & \text{それ以外の場合}\end{cases}$$

2文字の演算子 '& と'!は

A'& B='(A & B)

A'!B='(A!B)

と定義される.

11.2.4.3 パターン照合演算子

式の構文規則

式1 〔'〕 ? パターン

パターン ::= | パターン子〔パターン子〕... | @式子 V パターン |

パターン子 ::= | 整数字列 | . | | 定文字列 | パターンコード |

パターンコード ::= | C | N | P | A | L | U | E | ...

値の計算　　S '? P は '(S?P) と同じ値を持つと定義される.

パターンは1つかそれ以上のパターン子が連結したものである. nをパターンにおけるパターン子の個数とおく. S?パターンはSをn個の副文字列に部分化できるときに限り真であると言い

S=S1␣S2 . . . Sn

において Si とパターン子の間には，1対1の順序を保った対応関係があり，Si がそれぞれ対応するパターン子を“満足する”．Si のあるものは空文字列でもあることに注意．

各パターン子は2つの部分から成り，Si と比較をするもととなる1つのパターン コードまたは定文字列[訳注1)]と，それに先行する乗数である．後者は正確に反復回数を示す数字，あるいは“不定乗数”としての“．”である．

整数字列という形の乗数が存在する場合は，Si が整数字列個の副文字列を連結したものであり，それらの副文字列が指定された定文字列ないし，パターン子のパターン コードを満足するときに限り，Si がパターン子を満足すると言う．

もし不定乗数を示す“．”が存在する場合は，Si が何個（ゼロを含む）あってもよい副文字列を連結したものであり，これらの副文字列が定文字列ないしパターン子のパターン コードを満足するときに限り，Si がパターン子を満足すると言う．

副文字列はそれが定文字列の値に一致しているときに限りその定文字列を満足すると言う

パターン コードは符号の列であり，それらは各々次のような文字の分類を表わしている．

符号	文字の種類
C	DEL を含む33個の制御文字

訳注 1)　12.2 定文字列参照．

N	10個の数字
P	SP を含む33個の句読文字
A	52個のアルファベット全文字
L	26個のアルファベット小文字
U	26個のアルファベット大文字
E	すべて（文字の全集合）

パターン コードによって表わされる文字分類は，各パターン コードによって代表される文字集合の種類である．１つの副文字列はその中の文字がすべてパターン コードの代表する文字の種類に入るときに限りパターン コードを満足すると言う．

11.2.4.4 真偽値演算子の例

<u>関係演算子および論理演算子</u>

X=3, Y="A" とすると

式	値
1=1	1
1=Ø	Ø
(X=3)!(1=1)	1
(X=3)+1	2
X=3+1	2
X>Y	1
X>Y=1	1
Y'<X=1	Ø
Y=3!(Y="A")	1
Y="A" _ "B"	1B
"MGH"]Y<3	1

パターン照合演算子

初めの条件	式	値
SET␣X="123456"	X?.N	1
	X?2N4E	1
	X?1"12"4N	1
	X?.A	0
	X?.N1"6"	1
	X?.E1"3".N	1
SET␣X="A12B12C"	X?.N	0
	X?.NA	1
	X?2NA.E1"1".E	1
SET␣X="TESTTESTtest"	X?2"TEST".E	1
	X?1"TEST".E	1
	X?8U4L	1
	X?8A1"test"	1
	X?.U1"test"	1
	X?.A2L	1
SET␣X="12.34␣dollars"	X?.N1P.E	1
	X?2N1P2N1"␣".L	1
SET␣X=12.34	X?.N	0
SET␣X=1234.	X?.N	1
SET␣X="1234."	X?.N	0
SET␣X=$C(7)_$C(10)	X?2C	1
	X?.E1C.E	1

第 12 章
式　　　　子

12.1　式子の構文規則

式子とは，式値を生み出す 構成要素である．（式における 式子の役割に関しては，11.1.1参 照.）式子は実行されたときに１つの値を生ずる．この章では，式子の各々の形の構文規則と値について解説する．

式子の構文の定義は次のとおりである．右欄に式子の各形の詳細な解説の前後参照がある．

			この本で定義された場所
式子	::=	（式）	12.1.1
		特殊変数	第９章
		関数	第10章
		定文字列	12.2
		数字列	12.3
		記憶域参照	12.4
		'真偽値式子	12.5
		＋ / － 数式子	12.5

12.1.1　式子形成のための括弧の使用

式として表現できる計算手順は すべてそれを（式）と 書くことで，式子としてその値を計算できる．１つの式の構成要素である式子の値は，式の値の計算の前に計算されなければならないから，この括弧の使用は１つの式の構成要

素の数値を求めることの順序を制御する基礎なのである.（11.1.4ではこの問題についてなお検討を加え1つの例を示してある.）

12.2 定 文 字 列

“定字列”とはその値がそれのつづりのみからなる関数である式を言う. 定字列には2つの型があり，1つは（ここで述べられる）定文字列であり，他の1つは（12.3で述べられる）数字列である.

定文字列はその最初と最後に引用符をおくことによって示される. 定文字列の構文規則を定義するには，引用符以外の文字の集合を定義する必要がある. ここで定義を略式に述べれば，引用符以外の文字とは ASCII 文字の中から引用符（“）を除いたすべての文字である.

定文字列 ::= “ { “” | 引用符以外の文字 } … ”

要するに定文字列は引用符(これも定文字列の一部と考えられる)によって境界づけられ，その引用符の内側にいかなる ASCII の文字と符号の定字列を入れてもよいが，その引用符内で文字としての引用符を表わすには2個の引用符を続ける. 定文字列の場合, その値はそのつづりそのものであるが, 次のような除外がなされる.

1. 2つの境界の引用符はどちらも定文字列の値から除外される.
2. 引用符2つが一組となって境界引用符の内側にあれば，その2つのうち1つの引用符が削除される.

空文字列を示す定文字列は“”と書かれる.

12.3 数　字　列

(文脈上式子として適当とみなされる)数字列は最初の文字が数字または小数点であることによって示される. 数字列は定字列の一形であり本来その値が Vn に限定された算術計算用に用意されたものである.

数字列の構文規則は，それに前置符号が許されない点を除けば指数値の構文規則と同じものである．(11.1.3参照.)

数字列 ::= 仮数〔指数〕

数字列の値は，解釈演算 Ien (11.1.3.2参照) を直接に応用して，そのつづりから得られる．

整数字列という構成は，この本の他の構文定義の中で用いられている．

整数字列 ::= 数字〔数字〕...

整数字列の値は，解釈演算 Ii (11.1.3参照) を直接に応用して，そのつづりから得られる．

定字列の例

つづり	値
"HELLO"	HELLO
""""HELLO""""	"HELLO"
1	1
ØØØ1	1
1.23	1.23
1E1	1Ø
1E2	1ØØ
1.5E2	15Ø
3.1415E3	3141.5
2.34E−2	.Ø234
.Ø1E2	1
"ØØØ1"	ØØØ1

12.4 記憶域参照

記憶域参照は「名前付きパーティション記憶域」および「名前付きシステム記憶域」の各節に独特な指定子である．記憶域参照は式子以外の文脈内でも現われる――例えば $DATA 関数の引数，あるいは SET 命令の ＝ の左側で――ことがある．記憶域参照が式子の文脈内に現われるときは，常にその値は，指定された記憶節の「値」属性の内容である．この意味においてもし記憶域参照によって指定されたとおりの記憶節がないならば，あるいは記憶節があっても，それの「値」属性が定義されてないならば，記憶域参照の実行はエラーであると考えられそれは定義された値を持たない．

構文規則上，記憶域参照には以下の構文定義の中で挙げているように3つの変法がある．

名前　::=　| % / アルファ |　[数字 / アルファ] ...

〔Port: 移送基準に従ったルーチンでは名前の中にいかなる小文字も用いてはならない．9文字以上の文字を持つ名前では，最初の8文字によって識別されなければならない．　:Port〕

記憶域参照　::=　| ローカル変数 / グローバル変数 / 略式グローバル参照 |

ローカル変数　::=　| 名前〔(式〔, 式〕...)〕 / @式子 V ローカル変数 |

グローバル変数　::=　| ^名前〔(式〔, 式〕...)〕 / @式子 V グローバル変数 |

略式グローバル参照　::=　| ^(式〔, 式〕...) / @式子 V 略式グローバル参照 |

括弧内の式は“添字”と呼ばれる.

構文形	節指定場所
ローカル変数	名前付きパーティション記憶域
グローバル変数	名前付きシステム記憶域
略式グローバル参照	名前付きシステム記憶域

LOCK 命令は，略式グローバル参照でない名前のみの参照を必要とする．変数名がこのために存在する.

変数名 ::= | ローカル変数
 グローバル変数 |

記憶域参照の場合に記憶節を指定する演算法は，次の2つの段階から成る.

1. この本で「節指定マッピング」(次に述べる) と呼ばれる記憶域参照をその引数とするマッピングが実行される．このマッピングの値が節指定子となる (4.2 参照).
2. 計算された節指定子と同一の文字列を「名前」属性としてもつ記憶節が存在する場合には，それが指定された記憶節そのものである．もしそのような記憶節が存在しないならば，その結果としての作動は記憶域参照が現われる文脈次第である.

節指定マッピングの解説

「節指定 マッピング」とは空の「節指定 レジスター」によって始まり，その「節指定レジスター」の内容に右へ右へと連結してゆくことによって最終的に節指定子を作り上げることである.

1. 「節指定子レジスター」を“NDR”と呼ぶ．第1段階で行なうことは記憶域参照もしくは変数名の構文形式が何であるかに依存する.
   a. もしローカル変数ならば NDR にそれの名前を入れる.
   b. もしグローバル変数ならば，NDR にそれの^名前を入れる．(最初の

文字は "^" である.)

c. もし略式グローバル参照ならば NDR に「略式グローバル指標」の内容を入れる. もし「略式グローバル指標」が定義されていないならば, その場合の記憶域参照は誤りである.

2. もしその記憶域参照もしくは変数名が, 添子を含まないローカル変数もしくはグローバル変数であれば, この「節指定マッピング」はここで終了する. そうでなければ第3段階に進む.

3. 左から右への方向で, 各々の添字式について以下の a. b. を1回ずつ実行する.

a. まず式を評価しその値を V とする.

b. 次に ¢V を NDR の右に連結させる.

4. これで「節指定マッピング」が完了する. 引数が記憶参照である場合には, グローバル変数もしくは略式グローバル参照の実行が「略式グローバル指標」に及ぼす効果についての 5.2 の説明を参照するとよい. もし引数が変数名であるならば, 「略式グローバル指標」には何の効果もない.

〔Port: 移送基準に従ったルーチンでは, その値が負でない整数の構文規則をとる添字しか実行させてはならない.

負でない整数 ::= | Ø
 Øでない数字〔数字〕... |

添字式の値は最大9けたが許される. 各言語装備には9けた以下の負でない整数以外の値をとる添字式がきた場合のインタプリタの応答について説明がついているべきである. このような添字値を受け入れることのできるインタプリタは, この本の以下の部分との一貫性が保たれるような説明がついていなければならない. :Port〕

正しいローカル変数の例

X　AB　A(3)　AB(4, 5, X)　A(B(3))

A(3, 4)　AB(X, Y(Z), ^A(3, 4, 5))　A(B(C(D(E, F(G)))))

A(1)　A(−−1)　A(+1)　A("1")　A(1.∅)

A(∅1)　A(1.5E1)　A(1.)　A(+"12A")

移送性のない記憶域参照の例

A(1.5)　A(−1)　A("∅1")　A(12A)　A("12A")

A(1.55E1)　A("1.")

「略式グローバル参照」の例

「略式グローバル指標」の初めの内容	記憶域参照	節指定子	結果としての「略式グローバル指標」の内容
定義されていない	^A(1)	^A¢1	^A
^A	^(2)	^A¢2	^A
^A	^(3, 4)	^A¢3¢4	^A¢3
^A¢3	^(5)	^A¢3¢5	^A¢3
^A¢3	^B(3, 5)	^B¢3¢5	^B¢3
^B¢3	^X	^X	定義されていない
定義されていない	^(2)	誤	定義されていない
定義されていない	^Y	^Y	定義されていない
定義されていない	^X(5, 6, 7, 8)	^X¢5¢6¢7¢8	^X¢5¢6¢7
^X¢5¢6¢7	^(9)	^X¢5¢6¢7¢9	^X¢5¢6¢7
^X¢5¢6¢7	^(9, −2)	^X¢5¢6¢7¢9¢−2 (*)	^X¢5¢6¢7¢9 (*)
^X¢5¢6¢7¢9	^("+")	^X¢5¢6¢7¢9¢+ (*)	^X¢5¢6¢7¢9 (*)

(*) 注意： ある言語装備では当然これらを誤りとするであろう．誤りとしない場合はこれらが結果として現われるはずである．

12.5 一 項 演 算 子

3つの演算子 '(否定), +(正符号), および −(負符号) は一項演算子と呼ばれる. これらの効果は次のように定義される.

'真偽値式子の値は, 真偽値式子の値の補数である. つまり T を真偽値式子とすると

$$'T=\begin{cases}1 & \text{Tの値がØの場合}\\ Ø & \text{Tの値が1の場合}\end{cases}$$

+数式子の値は, 数式子の値と同じである. すなわち+は単に数解釈 In の実行を強いる手段にすぎない.

−数式子の値は, 数式子の値を負にしたものである. V を数式子の値とすれば

−数式子の値は, 文字列 −V に In を適用した結果である. (二重の負を単純化するには 11.1.3.1 の符号の省略規則を適用する.)

一項演算子を3つの一項演算子のどれを指してもよいと仮定すると, 一項演算子の参照となる式子の構文解説にあたる部分を次のように示すことができる.

式子 ::= 一項演算子 式子

この循環型の式表示は, 次の反復型の非公式の式表示

〔一項演算子〕. . . 式子

でもよいのであるが前者は一項演算子が右から左に評価されてゆくのを明確にしている. 従ってこれが公式として優先される.

一項演算子の例

式	値	
+“A”	Ø	
+−“A”	Ø	
−“5.Ø”	−5	
'“5.Ø”	Ø	
'(1+2E−1Ø−1)	1	(多分)[訳注1)]
'(1−1+2E−1Ø)	Ø	(決定的に)[訳注2)]
3−−5	8	
3−Ø−−5	8	
3−−−5	−2	
3−−'“A”	4	

訳注 1) 1+2E−1Ø の結果は 1.0000000002 であるがこれは9けたを超えるのでけた落ちして1となり，これと −1 とで Ø，それを否定して1となるであろう．しかしけた落ちしない言語装備があるかも知れない．

訳注 2) 1−1 で Ø，これに 2E−1Ø を加えると Ø.0000000002，これは有効数字が1けたなのでけた落ちしない．従ってこれを否定して Ø となる．

付録 A

ASCII 文字表

文字表記は，ANS X3.4-1968に規定されたものである．そのコード値は，$ASCII関数の値として，また，$CHAR関数の引数として現われるものである．

Code	Character	Code	Character	Code	Character	Code	Character
Ø	NUL	32	SP	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	1ØØ	d
5	ENQ	37	%	69	E	1Ø1	e
6	ACK	38	&	7Ø	F	1Ø2	f
7	BEL	39	'	71	G	1Ø3	g
8	BS	4Ø	(	72	H	1Ø4	h
9	HT	41	)	73	I	1Ø5	i
1Ø	LF	42	*	74	J	1Ø6	j
11	VT	43	+	75	K	1Ø7	k
12	FF	44	,	76	L	1Ø8	l
13	CR	45	−	77	M	1Ø9	m
14	SO	46	.	78	N	11Ø	n
15	SI	47	/	79	O	111	o
16	DLE	48	Ø	8Ø	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	5Ø	2	82	R	114	r
19	DC3	51	3	83	S	115	s
2Ø	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	12Ø	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	9Ø	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	6Ø	<	92	\	124	¦
29	GS	61	=	93	]	125	}
3Ø	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

all plus 2

付録 B

索　　　引

（原書では構文規則上の用語を付録Bとし，システム モデル上の技術用語を付録Cとしているが，邦訳版ではこれらを合併しさらに，重要な用語若干を追加してある．）

␣　スペース (space)　8

¢　分離子 (separator)　15, 16

¢¢　分離子 (separator)　17, 18

@　間接指定呼び出し　7, 32

〔ア　行〕

「値」属性 (Value attribute)　14

後付け条件 (post-conditional)　39, 41

一項演算子 (unary operator)　164

移送基準 (Portability Requirement)　9

入口参照 (entry reference)　36

引数 (argument)　37, 108

引数並記 (argument list)　40, 108

引用符以外の文字 (nonquote)　158

英字 (alpha)　8

「Open リスト」(Open List)　11, 18

「Open リスト P-ベクトル」(Open List P-vector)　11, 13, 18

〔カ　行〕

解釈演算 (interpretation operations)　137

　指数解釈 (Ie)　139, 141

　　指数-数解釈 (Ien)　139, 141

　真偽値解釈 (It)　139

　数解釈 (In)　139

　数-真偽値解釈 (Int)　139, 142

　数-整数解釈 (Ini)　139, 142

　整数解釈 (Ii)　139

階層 (level)　13

下位の (descendant)　16

　直接下位の (immediate descendant)　15

「下位排他規則」(Descendant Exclusivity Rule)　17

仮数 (mantissa)　139

空文字列 (empty string)　158

関係演算子 (relational operator)　146

関数 (function)　110

間接指定 (indirection)　7, 32, 42

記憶域参照 (storage reference)　160

行 (line)　34

行参照 (line reference)　35

行体 (line body)　34

行頭 (line head)　34

「行バッファ」(Line Buffer)　27

「行ポインタ」(Line Pointer)　27

グローバル変数 (global variable)　160

言語装備 (implementation)　1

言語装備者 (implementor)　2

「現用装置指定子」(Current Device Designator)　21, 23

「現用装置水平カーソル」(Current Device Horizontal Cursor)　21, 25

「現用装置垂直カーソル」(Current Device Vertical Cursor)　21, 25

「現用装置占有規則」(Current Device Ownership Rule)　24

構文規則上の用語型 (Syntactic Type)　3

〔サ　行〕

「作業番号」(Job Number)　11, 19
「作業番号 P-ベクトル」
(Job Number P-vector)　11, 13, 19
算術演算子 (arithmetic operator)　145
時間制限 (timeout)　44
式 (expression)　133
式子 (expression atom)
133, 143, 157, 164
式尾 (expression tail)　133, 143, 145
指数 (exponent)　139
指数式 (exponential expression)　138
指数式子 (exponential expression atom)　138
指数値 (exponential value)　139
「システム記憶域」(System Storage)　10
「主装置」(Principal Device)　24
「主装置慣例」(Principal Device Convention)　24
終結手順 (termination procedure)　78
上位の (ascendant)　16
直接上位の (immediate ascendant)　15
小数値 (fraction value)　136
書式 (format)　43
真偽演算子 (truth operator)　146
真偽値 (truth value)　135
真偽値式 (truth-value expression)　138
真偽値式子 (truth-value expression atom)　138
数字 (digit)　136
数式 (numeric expression)　138
数式子 (numeric expression atom)　138
数字列 (numeric literal)　159
数字連鎖 (digit sequence)　139
数値 (numeric value)　136
整数式 (integer expression)　138
整数式子 (integer expression atom)　138
整数字列 (integer literal)　159
整数値 (integer value)　136
節 (node)　13
節指定子 (node designator)　16
「節指定マッピング」
(Node Designation Mapping)　161
「節指定レジスター」
(Node Designator Registor)　161
0でない数字 (nonzero digit)　136
装置指定子 (device designator)　18
装置指定式 (device specifier)　72
「装置排他規則」(Device Exclusivity Rule)　18
装置パラメータ (device parameter)　72
初値-増分パラメータ (start-step parameter)　52
属性ブロック (attribute block)　14

〔タ　行〕

単元操作 (single atomic operation)　27
注釈文 (comment)　34
通常の実行順序 (nomal execution sequence)　27
定文字列 (string literal)　158
定字列 (literal)　158
「D」属性 (D attribute)　14
d ラベル (d label)　35
「テスト スイッチ」(Test Switch)　21, 22
登録節 (directory node)　13
特殊変数 (special variable)　97, 98
「時計」(Clock)　20
「時計レジスタ」(Clock Registor)　13

〔ナ　行〕

名前 (name)　160
「名前」属性 (Name attribute)　14, 16

「名前付きパーティション記憶域」
(Named Partition Storage)　21
「名前付きシステム記憶域」
(Named System Storage)　13
二項演算子 (binary operator)　133

〔ハ　行〕

パターン (pattern)　153
パターン子 (pattern atom)　153
パターン コード (pattern code)　153
パーティション (partition)　10
「パーティション スタック」
(Partition Stack)　11, 27
「パーティション記憶域」
(Partition Storage)　11
パーティション スペース
(partition space)　102
「パラメータ マーカ」
(Parameter Marker)　30
「範囲 マーカ」(Scope Marker)　31
負でない整数 (nonnegative integer)　162
ヘッド (head)　141
変数名 (variable name)　161
for パラメータ (for parameter)　52
「For 範囲」スイッチ
(For Scope switch)　27

〔マ　行〕

MUMPS 値 (MUMPS value)　135
命令 (command)　37
命令語 (command word)　38
文字と符号 (graphic)　8
「文字ポインタ」(Character Pointer)　28
文字列演算子 (string operator)　145

〔ラ　行〕

ラベル (label)　34
「略式グローバル指標」
(Naked Indicator)　21
略式グローバル参照 (naked
reference)　160
ルーチン (routine)　33
ルーチン参照 (routine reference)　36
ルーチン体 (routine body)　33
ルーチン頭 (routine head)　33
ルーチン名 (routine name)　33
ローカル変数 (local variable)　160
「Lock リスト」(Lock List)　11, 16
「Lock リスト P-ベクトル」
(Lock List P-vector)　10, 13, 16
論理演算子 (logical operator)　146

付録 C

MUMPS 開発委員会文書

MDC 文書番号	内容
NBS Handbook 118	January 1976 (with errata sheets thru March 9, 1976), MUMPS Language Standard
	Part Ⅰ: MDC/28, 3/12/75, MUMPS Language Specification M.E. Conway
	Part Ⅱ: MDC/33, 9/17/75, MUMPS Transition Diagrams D.D. Sherertz and Anthony I. Wasserman
	Part Ⅲ: MDC/34, 9/17/75, MUMPS Portability Requirements E.A. Gardner and C.B. Lazarus
29	5/28/75, MUMPS Interpreter Validation Program User Guide J. Rothmeier and P.L. Egerman
30	6/25/75, MUMPS Translation Methodology P.L. Egerman, C.B. Lazarus and P.T. Ragon
35	11/10/76, MUMPS Documentation Manual L.J. Peck and R.A. Greenes
1/11	6/13/75, MUMPS Primer M.E. Johnson and R.E. Dayhoff
2/1	5/15/75, MUMPS Globals and Their Implementation A.I. Wasserman, D.D. Sherertz and C.L. Rogerson
2/2	5/30/75, Design of Multiprogramming System for the MUMPS Language A.I. Wasserman, D.D. Sherertz and R.W. Zears
2/3	6/15/75, Implementation of the MUMPS Language Standard A.I. Wasserman and D.D. Sherertz
3/5	8/31/76, MUMPS Programmers' Reference Manual M.E. Conway and P.L. Egerman

〒460 名古屋市中区新栄 2丁目1-9 雲竜ビル東館
マンプス システム研究所内
日本マンプス ユーザーズ グループ事務局 TEL (052) 251-7408(代)

邦訳付録

翻訳にあたって用いたMUMPSの文字列の分類と訳語

標準 MUMPS の全文字列概念

- データ（ASCII 文字の１つ又はいくつかを連結したもの，又は何もなし（空文字列 empty string））
  - ↑（$TEXT）
  - ↓（間接指定 XECUTE）
- プログラムコード
  - 式以外のもの……命令語（command word），ラベル（label），注釈文（comment），etc.
  - 式（expression）
    - 値をもたず式子と式子の関係を与えるもの：二項演算子（binary operator）……+，−，*，／，＼，#，>，<，]，[，=，?，&，!，etc.
    - 値をもつもの：式子（expression atom）（一項演算子（unary operator +，−，'）を含む）
      - 値が変化するもの
        - 記憶域参照（storage reference）
        - 関数（function）
        - 特殊変数（special variable）
      - 値が変化しないもの　定字列(literal)
        - 定文字列（string literal）引用符ではさまれた文字列（空文字列を表わす場合は引用符を２つ続ける）
        - 数字列（numeric literal）
          - 整数字列（integer literal）
          - 整数字列以外のもの……小数，指数表示

訳者の了解
により検印省略

標準マンプス 言語マニュアル

昭和52年11月30日 初版印刷
昭和52年12月15日 初版発行

定価 2800 円

訳者 若井一朗（わかいいちろう）
名古屋市中区新栄2丁目1-9
田中豊穂（たなかとよほ）
岐阜市長良宮路町1-17
嶋芳成（しまよしなり）
春日井市篠木町8-2802

発行者 株式会社 コロナ社
代表者 牛來武知
東京都文京区千石4-46-10

印刷者 有限会社 大和製版印刷所
代表者 田中健
東京都荒川区西日暮里2-48-2

112 東京都文京区千石 4-46-10
発行所 株式会社 コロナ社
CORONA PUBLISHING CO., LTD.
Tokyo Japan
振替 東京 4-14844・電話(03)941-3131(代)

3050-100200-:353 (染野製本所)